Binomial Checkpointing for Arbitrary Programs with No User Annotation

Jeffrey Mark Siskind, qobi@purdue.edu

PURDUE UNIVERSITY

CSE2017, Tuesday 28 February 2017

Joint work with Barak Avrum Pearlmutter

Automatic Differentiation (AD)

$$f = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1$$

Automatic Differentiation (AD)

$$
\begin{aligned}
f &= f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1 \\
f'(x_1) &= f'_t(x_t) \times f'_{t-1}(x_{t-1}) \times \cdots \times f'_2(x_2) \times f'_1(x_1)
\end{aligned}
$$

Automatic Differentiation (AD)

$$
\begin{aligned}
f &= f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1 \\
f'(x_1) &= f_t'(x_t) \times f_{t-1}'(x_{t-1}) \times \cdots \times f_2'(x_2) \times f_1'(x_1) \\
f'(x_1)^\top &= f_1'(x_1)^\top \times f_2(x_2)^\top \times \cdots \times f_{t-1}'(x_{t-1})^\top \times f_t'(x_t)^\top
\end{aligned}
$$

$$
\begin{aligned}
f &= f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1 \\
f'(x_1) \times \acute{x}_1 &= f_t'(x_t) \times f_{t-1}'(x_{t-1}) \times \cdots \times f_2'(x_2) \times f_1'(x_1) \times \acute{x}_1 \\
f'(x_1)^\top \times \grave{y} &= f_1'(x_1)^\top \times f_2(x_2)^\top \times \cdots \times f_{t-1}'(x_{t-1})^\top \times f_t'(x_t)^\top \times \grave{y}
\end{aligned}
$$

Automatic Differentiation (AD)

$$f = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1$$

$$x_2 = f_1(x_1)$$

Automatic Differentiation (AD)

$$f = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1$$

$$x_3 = f_2(f_1(x_1))$$

Automatic Differentiation (AD)

$$f = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1$$

$$x_t = \quad f_{t-1}(\ldots f_2(f_1(x_1))\ldots)$$

Automatic Differentiation (AD)

$$f = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1$$

$$y = f_t(f_{t-1}(\ldots f_2(f_1(x_1))\ldots))$$

Automatic Differentiation (AD)

$=$ $f_1'(x_1) \times \acute{x}_1$

x_1

Automatic Differentiation (AD)

$$= \quad f_2'(x_2) \times f_1'(x_1) \times \acute{x}_1$$

$$x_2 = f_1(x_1)$$

Automatic Differentiation (AD)

$$
\begin{aligned}
&= \quad f'_{t-1}(x_{t-1}) \times \cdots \times f'_2(x_2) \times f'_1(x_1) \times \acute{x}_1 \\
x_{t-1} &= \quad f_{t-2}(\ldots f_2(f_1(x_1))\ldots)
\end{aligned}
$$

Automatic Differentiation (AD)

$$f'(x_1) \times \acute{x}_1 = f_t'(x_t) \times f_{t-1}'(x_{t-1}) \times \cdots \times f_2'(x_2) \times f_1'(x_1) \times \acute{x}_1$$

$$x_t = \quad f_{t-1}(\ldots f_2(f_1(x_1)) \ldots)$$

Automatic Differentiation (AD)

x_1

x_1

Automatic Differentiation (AD)

$$x_2 = f_1(x_1)$$

$$x_2, x_1$$

Automatic Differentiation (AD)

$$x_{t-1} = f_{t-2}(\ldots f_2(f_1(x_1))\ldots)$$

$$x_{t-1}, \ldots, x_2, x_1$$

Automatic Differentiation (AD)

$$x_t = \quad f_{t-1}(\ldots f_2(f_1(x_1))\ldots)$$

$$x_t, x_{t-1}, \ldots, x_2, x_1$$

Automatic Differentiation (AD)

$$f_t'(\color{red}{x_t})^\top \times \grave{y}$$

$$\color{red}{x_t}\color{black}{, x_{t-1}, \ldots, x_2, x_1}$$

Automatic Differentiation (AD)

$$f'_{t-1}(x_{t-1})^\top \times f'_t(x_t)^\top \times \grave{y}$$

$$x_{t-1}, \ldots, x_2, x_1$$

Automatic Differentiation (AD)

$$f_2(x_2)^\top \times \cdots \times f'_{t-1}(x_{t-1})^\top \times f'_t(x_t)^\top \times \grave{y}$$

x_2, x_1

Automatic Differentiation (AD)

$$f'(x_1)^\top \times \grave{y} = f_1'(x_1)^\top \times f_2(x_2)^\top \times \cdots \times f_{t-1}'(x_{t-1})^\top \times f_t'(x_t)^\top \times \grave{y}$$

x_1

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors
- computing ∇f with reverse mode requires a tape with t entries

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors
- computing ∇f with reverse mode requires a tape with t entries
- computing ∇f with forward mode entails an increase of $O(n)$ time but $O(1)$ space over computing f

Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors
- computing ∇f with reverse mode requires a tape with t entries
- computing ∇f with forward mode entails an increase of $O(n)$ time but $O(1)$ space over computing f
- computing ∇f with reverse mode entails an increase of $O(t)$ space but $O(1)$ time over computing f

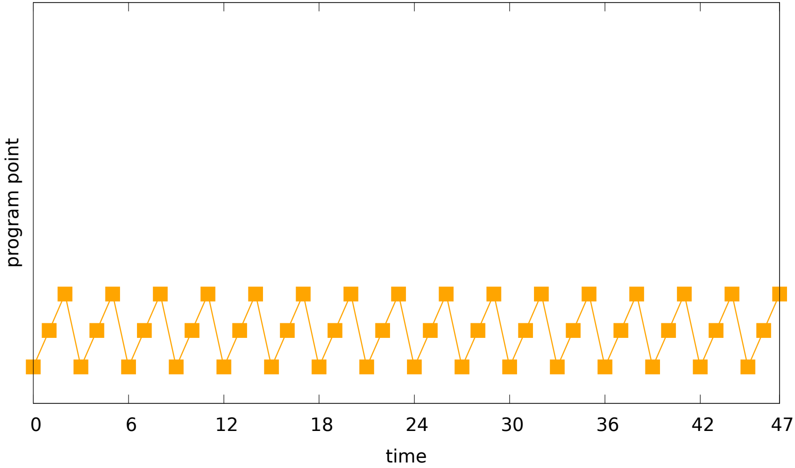
Typical Case

- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors
- computing ∇f with reverse mode requires a tape with t entries
- computing ∇f with forward mode entails an increase of $O(n)$ time but $O(1)$ space over computing f
- computing ∇f with reverse mode entails an increase of $O(t)$ space but $O(1)$ time over computing f
- both n and t are large

Typical Case

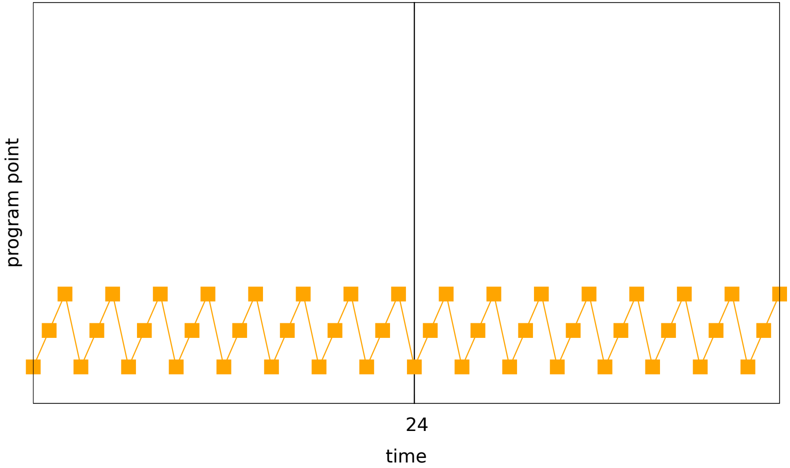
- $f : \mathbb{R}^n \to \mathbb{R}$
- Jacobian is $1 \times n$, aka ∇f
- computing ∇f takes n calls to forward mode with $\acute{x}_1$ being basis vectors
- computing ∇f with reverse mode requires a tape with t entries
- computing ∇f with forward mode entails an increase of $O(n)$ time but $O(1)$ space over computing f
- computing ∇f with reverse mode entails an increase of $O(t)$ space but $O(1)$ time over computing f
- both n and t are large
- today: computing ∇f with checkpointing reverse mode entails an increase of $O(\log n)$ time and $O(\log t)$ space over computing f

Execution Trace of Loop



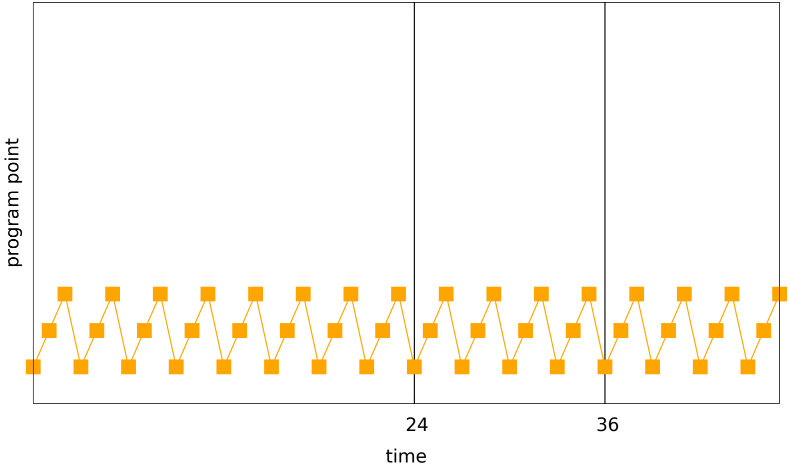
Execution Trace of Loop

Easy to make regular and uniform checkpoints



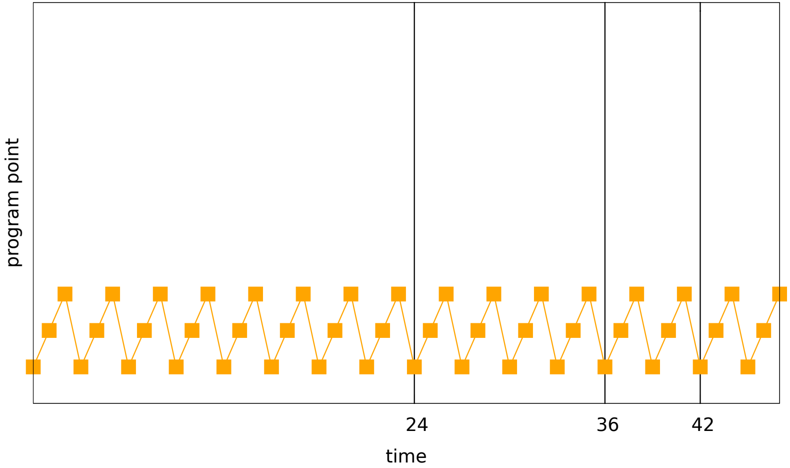
Execution Trace of Loop

Easy to make regular and uniform checkpoints



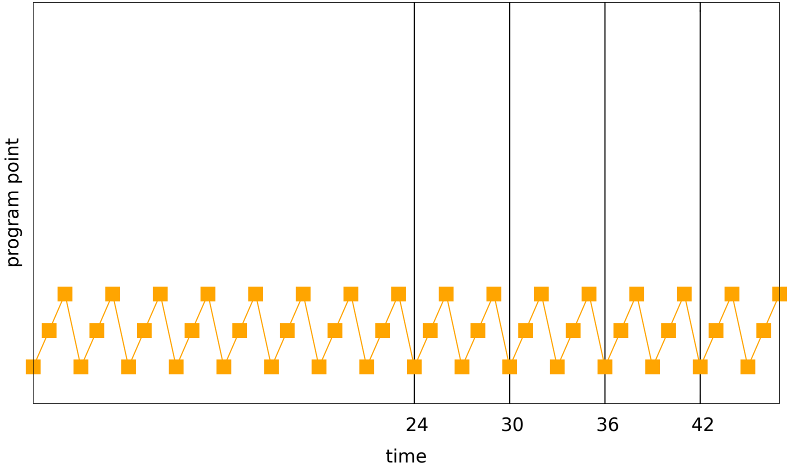
Execution Trace of Loop

Easy to make regular and uniform checkpoints



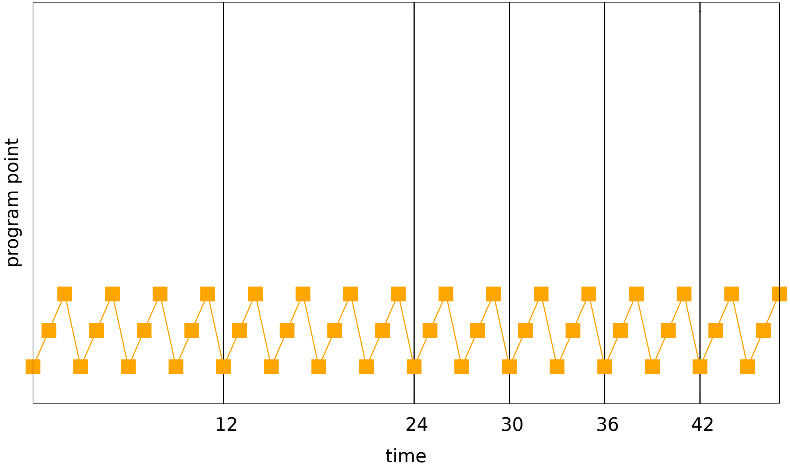
Execution Trace of Loop

Easy to make regular and uniform checkpoints



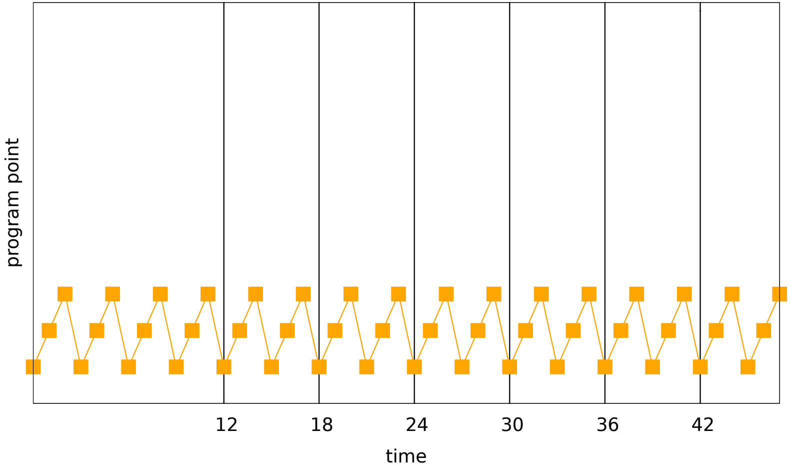
Execution Trace of Loop

Easy to make regular and uniform checkpoints



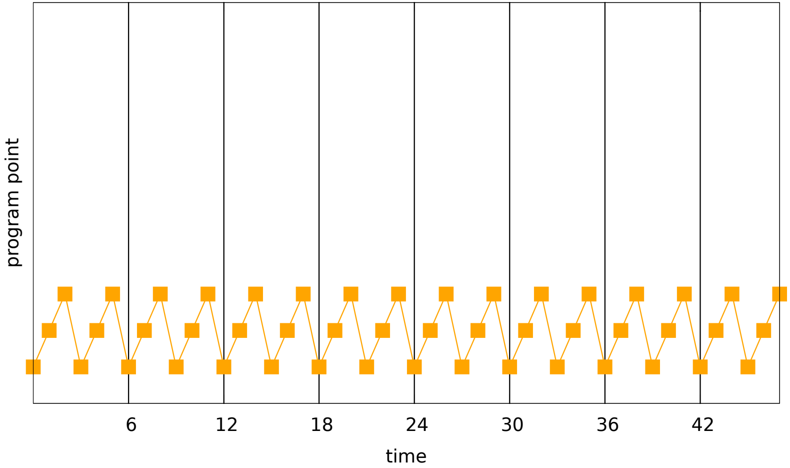
Execution Trace of Loop

Easy to make regular and uniform checkpoints

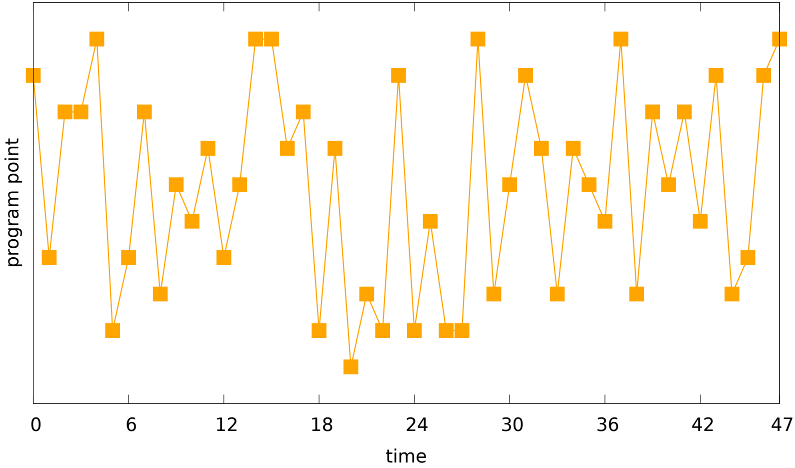


Execution Trace of Loop

Easy to make regular and uniform checkpoints

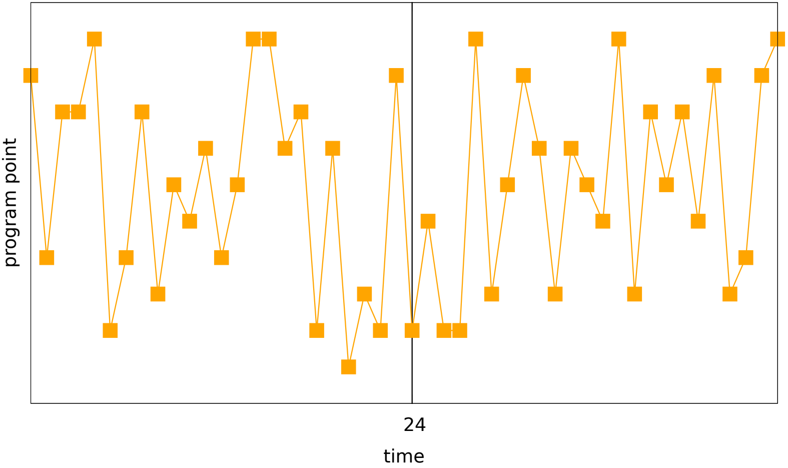


Execution Trace of Arbitrary Code



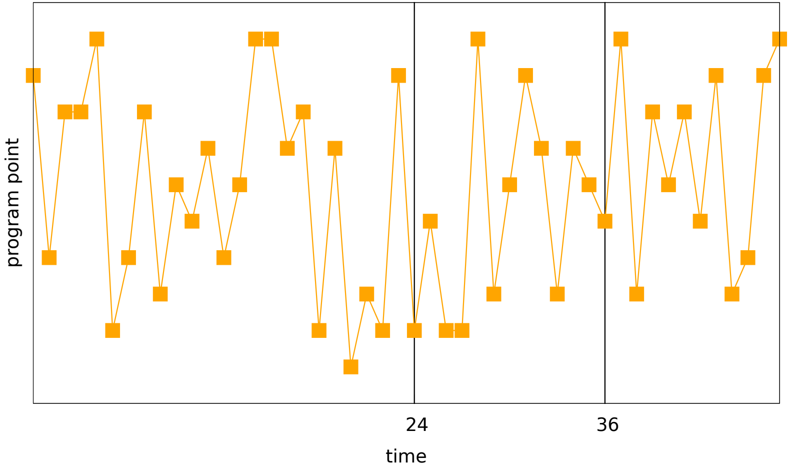
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



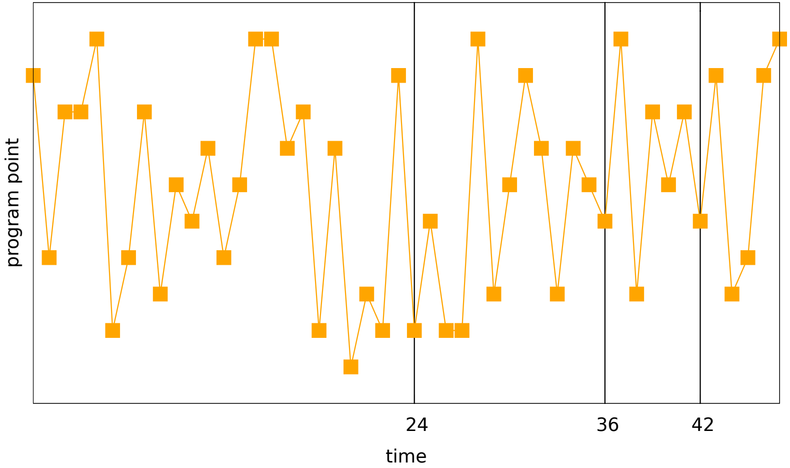
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



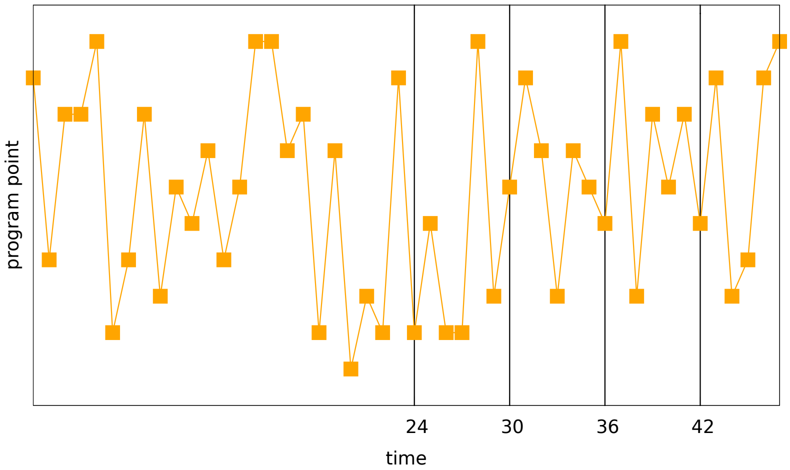
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



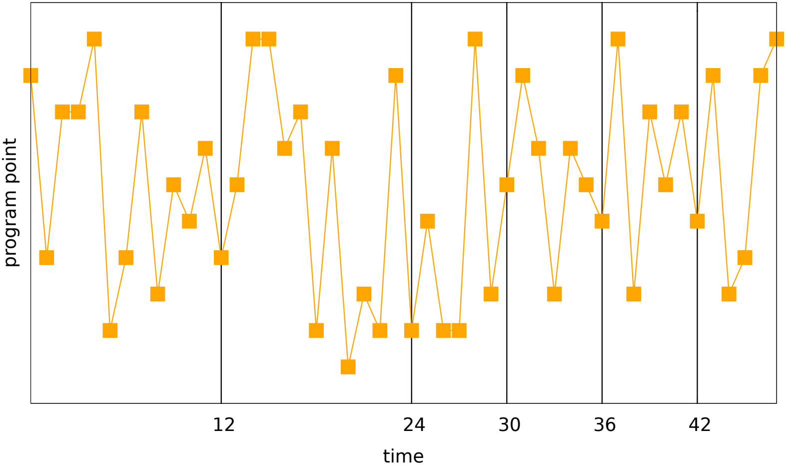
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



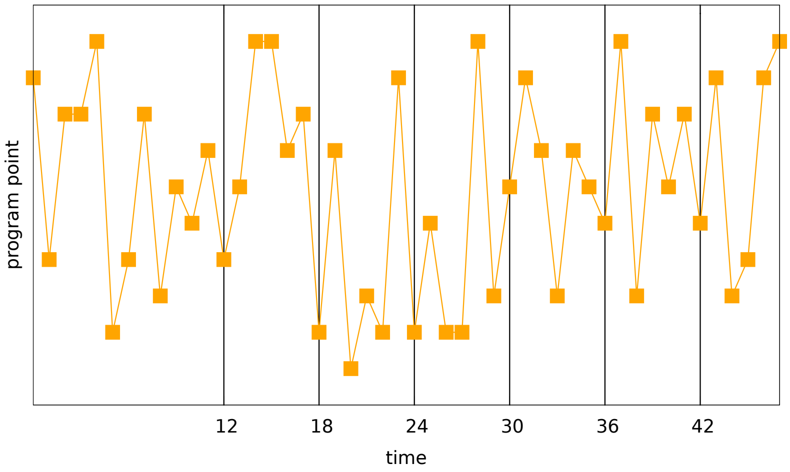
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



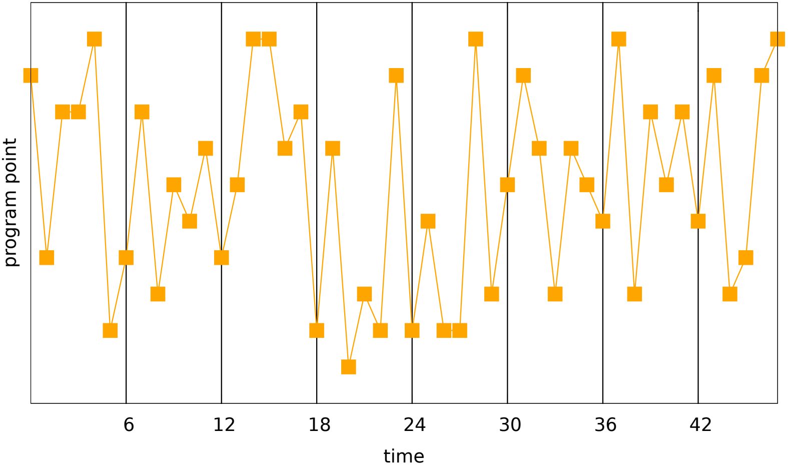
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



Key Idea

```
function main(w)
    local x = f(w)
    local y = h(g(x))
    local z = p(y)
    return z
end
```

$\leadsto$

```
function main(w)
    for i = 1, 5
    if i==1 then
        local x = f(w)
    elseif i==2 then
        local t = g(x)
    elseif i==3 then
        local y = h(t)
    elseif i==4 then
        local z = p(y)
    elseif i==5 then
        return z
    end
end
```

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

inductive case:

$$h \circ g = f \quad (1)$$
$$u = g\ x \quad (2)$$
$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \quad (3)$$
$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \quad (4)$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case (f x fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ {\color{red}f}\ x\ \grave{y}$:

$$\textbf{base case } (f\ x \text{ fast}): \quad (y, \grave{x}) = \overleftarrow{\mathcal{J}}\ {\color{red}f}\ x\ \grave{y} \tag{0}$$

$$\textbf{inductive case}: \quad h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \quad (0)$$

inductive case:
$$h \circ g = f \quad (1)$$
$$u = g\ x \quad (2)$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \quad (3)$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \quad (4)$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ \color{red}{g} = f \tag{1}$$

$$u = \color{red}{g}\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; f\; x\; \grave{y}$:

base case ($f\; x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\; f\; x\; \grave{y}$ (0)

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\; x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\; h\; u\; \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; g\; x\; \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$\color{red}{h} \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ \color{red}{h}\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast):
$$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast):

$$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Algorithm for Bisection Checkpointing

To compute $(y, \grave{x}) = \overset{\surd}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

inductive case:

$$h \circ g = f \quad (1)$$

$$u = g\ x \quad (2)$$

$$(y, \grave{u}) = \overset{\surd}{\mathcal{J}}\ h\ u\ \grave{y} \quad (3)$$

$$(u, \grave{x}) = \overset{\surd}{\mathcal{J}}\ g\ x\ \grave{u} \quad (4)$$

A General Checkpointing API

$\text{PRIMOPS}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\text{CHECKPOINT}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\text{RESUME}\ u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\text{PRIMOPS}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\text{CHECKPOINT}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\text{RESUME}\ u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT } f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\textsc{primops}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\textsc{checkpoint}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\textsc{resume}\ u \mapsto y$	If $u = (\textsc{checkpoint}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\textsc{primops}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\textsc{checkpoint}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\textsc{resume}\ u \mapsto y$	If $u = (\textsc{checkpoint}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

PRIMOPS $f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
CHECKPOINT $f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
RESUME $u \mapsto y$	If $u = (\text{CHECKPOINT}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\textsc{primops}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\textsc{checkpoint}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\textsc{resume}\ u \mapsto y$	If $u = (\textsc{checkpoint}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\textsc{primops}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\textsc{checkpoint}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\textsc{resume}\ u \mapsto y$	If $u = (\textsc{checkpoint}\ f\ x\ n)$, return $y = f(x)$.

A General Checkpointing API

$\textsc{primops}\ f\ x \mapsto (y, n)$	Return $y = f(x)$ along with the number n of steps needed to compute y.
$\textsc{checkpoint}\ f\ x\ n \mapsto u$	Run the first n steps of the computation of $f(x)$ and return a checkpoint u.
$\textsc{resume}\ u \mapsto y$	If $u = (\textsc{checkpoint}\ f\ x\ n)$, return $y = f(x)$.

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:

$$h \circ g = f \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} \tag{0}$$

inductive case:
$$h \circ g = f \tag{1}$$
$$u = g\ x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

$$\begin{aligned}
&\textbf{base case } (f\ x \text{ fast}): && (y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} && (0)\\
&\textbf{inductive case}: && (y, 2n) = \text{PRIMOPS}\ f\ x && (1)\\
& && u = g\ x && (2)\\
& && (y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} && (3)\\
& && (u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} && (4)
\end{aligned}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

inductive case:

$$(y, 2n) = \text{PRIMOPS}\ f\ x \tag{1}$$

$$u = g\ x \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; f\; x\; \grave{y}$:

base case ($f\; x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\; f\; x\; \grave{y} \tag{0}$$

inductive case:
$$(y, 2n) = \text{PRIMOPS}\; f\; x \tag{1}$$
$$u = \color{red}{g}\; x \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\; h\; u\; \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; g\; x\; \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, {\color{red}f}\, x\, \grave{y}$:

base case (f x fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\, f\, x\, \grave{y} \tag{0}$$

inductive case:

$$(y, {\color{red}2n}) = \text{PRIMOPS}\, f\, x \tag{1}$$

$$u = {\color{red}\text{CHECKPOINT}\, f\, x\, n} \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\, h\, u\, \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, g\, x\, \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

$$\begin{aligned}
&\textbf{base case } (f\ x \text{ fast}): && (y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} && (0)\\
&\textbf{inductive case}: && (y, 2n) = \text{PRIMOPS}\ f\ x && (1)\\
& && u = \text{CHECKPOINT}\ f\ x\ n && (2)\\
& && {\color{red}(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y}} && (3)\\
& && (u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} && (4)
\end{aligned}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

base case ($f\ x$ fast): $(y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

inductive case:

$$(y, 2n) = \text{PRIMOPS}\ f\ x \tag{1}$$

$$u = \text{CHECKPOINT}\ f\ x\ n \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ h\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, f\ x\ \grave{y}$:

base case ($f\ x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\, f\ x\ \grave{y} \tag{0}$$

inductive case:

$$(y, 2n) = \text{PRIMOPS}\ f\ x \tag{1}$$

$$u = \text{CHECKPOINT}\ f\ x\ n \tag{2}$$

$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda u.\text{RESUME}\ u)\ u\ \grave{y} \tag{3}$$

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, f\; x\; \grave{y}$:

base case ($f\; x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\, f\; x\; \grave{y} \tag{0}$$

inductive case:
$$(y, 2n) = \text{PRIMOPS}\; f\; x \tag{1}$$
$$u = \text{CHECKPOINT}\; f\; x\; n \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\, (\lambda u.\text{RESUME}\; u)\; u\; \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, g\; x\; \grave{u} \tag{4}$$

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; f\; x\; \grave{y}$:

base case ($f\; x$ fast): $$(y, \grave{x}) = \overleftarrow{\mathcal{J}}\; f\; x\; \grave{y} \tag{0}$$

inductive case:
$$(y, 2n) = \text{PRIMOPS}\; f\; x \tag{1}$$
$$u = \text{CHECKPOINT}\; f\; x\; n \tag{2}$$
$$(y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\; (\lambda u.\text{RESUME}\; u)\; u\; \grave{y} \tag{3}$$
$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\; g\; x\; \grave{u} \tag{4}$$

Bisection Checkpointing via General Checkpointing API

To compute $(y, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ f\ x\ \grave{y}$:

$$
\begin{array}{lll}
\textbf{base case } (f\ x \text{ fast}): & (y, \grave{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \grave{y} & (0) \\
\textbf{inductive case}: & (y, 2n) = \text{PRIMOPS}\ f\ x & (1) \\
& u = \text{CHECKPOINT}\ f\ x\ n & (2) \\
& (y, \grave{u}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda u.\text{RESUME}\ u)\ u\ \grave{y} & (3) \\
& (u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ (\lambda x.\text{CHECKPOINT}\ f\ x\ n)\ x\ \grave{u} & (4)
\end{array}
$$

CPS Conversion

```
function f(x)
   return q(p(g(x), h(x)))
end
```

$\leadsto$

```
function f(c, x)
   return g(function(t1)
               return h(function(t2)
                           return p(function(t3)
                                       return q(c, t3)
                                    end, t1, t2)
                        end, x)
            end, x)
end
```

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\,x \\
[\lambda x.e]_c & \rightsquigarrow & c\,\lambda c\,x.[e]_c \\
[e_1\,e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\; x \\
[\lambda x.e]_c & \rightsquigarrow & c\; \lambda c\; x.[e]_c \\
[e_1\; e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\; c\; x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_{\color{red}c} & \rightsquigarrow & c\;x \\
[\lambda x.e]_c & \rightsquigarrow & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_{\color{red}c} & \rightsquigarrow & {\color{red}c}\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \leadsto & c\ x \\
[\lambda x.e]_c & \leadsto & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \leadsto & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\;x \\
[\lambda x.e]_c & \rightsquigarrow & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{aligned}
[x]_c &\leadsto c\,x\\
[\lambda x.e]_{\color{red}c} &\leadsto c\,\lambda c\,x.[e]_c\\
[e_1\,e_2]_c &\leadsto [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}}\\
e_0 &\leadsto [e_0]_{\lambda x.x}
\end{aligned}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\,x \\
[\lambda x.e]_c & \rightsquigarrow & c\,\lambda {\color{red}c}\,x.[e]_c \\
[e_1\,e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\; x \\
[\lambda x.e]_c & \rightsquigarrow & c\; \lambda c\; x.[e]_c \\
[e_1\; e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\; c\; x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \leadsto & c\;x \\
[\lambda x.e]_c & \leadsto & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \leadsto & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_{\color{red}c} & \rightsquigarrow & {\color{red}c}\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \leadsto & c\,x \\
[\lambda x.e]_c & \leadsto & c\,\lambda c\,x.[e]_c \\
[{\color{red}e_1\ e_2}]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}} \\
e_0 & \leadsto & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_{\color{red}{c}} & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$\begin{array}{lcl}
[x]_c & \leadsto & c\;x \\
[\lambda x.e]_c & \leadsto & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \leadsto & [e_0]_{\lambda x.x}
\end{array}$$

CPS Conversion is a Program Transformation

$$
\begin{aligned}
[x]_c &\rightsquigarrow c\,x\\
[\lambda x.e]_c &\rightsquigarrow c\,\lambda c\,x.[e]_c\\
[e_1\,e_2]_c &\rightsquigarrow [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}}\\
e_0 &\rightsquigarrow [e_0]_{\lambda x.x}
\end{aligned}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \leadsto & c\ x \\
[\lambda x.e]_c & \leadsto & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \leadsto & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\;x \\
[\lambda x.e]_c & \rightsquigarrow & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\;x \\
[\lambda x.e]_c & \rightsquigarrow & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ {\color{red}e_2}]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda {\color{red}x_2}.x_1\ c\ {\color{red}x_2}}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\; x \\
[\lambda x.e]_c & \rightsquigarrow & c\; \lambda c\; x.[e]_c \\
[e_1\; e_2]_{\color{red}{c}} & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\; {\color{red}{c}}\; x_2}} \\
e_0 & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\,x \\
[\lambda x.e]_c & \rightsquigarrow & c\,\lambda c\,x.[e]_c \\
[e_1\;e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\,c\,x_2}} \\
{\color{red}e_0} & \rightsquigarrow & [e_0]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \rightsquigarrow & c\ x \\
[\lambda x.e]_c & \rightsquigarrow & c\ \lambda c\ x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\ c\ x_2}} \\
e_0 & \rightsquigarrow & [{\color{red}e_0}]_{\lambda x.x}
\end{array}
$$

CPS Conversion is a Program Transformation

$$
\begin{array}{lcl}
[x]_c & \leadsto & c\;x \\
[\lambda x.e]_c & \leadsto & c\;\lambda c\;x.[e]_c \\
[e_1\;e_2]_c & \leadsto & [e_1]_{\lambda x_1.[e_2]_{\lambda x_2.x_1\;c\;x_2}} \\
e_0 & \leadsto & [e_0]_{\color{red}{\lambda x.x}}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lll}
[x]_c & \rightsquigarrow & c \qquad x \\
[\lambda x.e]_c & \rightsquigarrow & c \qquad \lambda c \quad x.[e]_c \\
[e_1\ e_2]_c & \rightsquigarrow & [e_1]_{(\lambda \quad x_1.[e_2]_{(\lambda \quad x_2.x_1\ c \quad x_2)} \quad)}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n} & \rightsquigarrow & c \quad n \qquad x \\
[\lambda x.e]_{c,n} & \rightsquigarrow & c \quad n \qquad \lambda c\ n \quad x.[e]_{c,n} \\
[e_1\ e_2]_{c,n} & \rightsquigarrow & [e_1]_{(\lambda n_1 \quad x_1.[e_2]_{(\lambda n_2 \quad x_2.x_1\ c\ n_2 \quad x_2),n_1} \quad),\ n}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n} & \rightsquigarrow & c\ (n+1)\ \ x \\
[\lambda x.e]_{c,n} & \rightsquigarrow & c\ (n+1)\ \ \lambda c\ n\ \ x.[e]_{c,n} \\
[e_1\ e_2]_{c,n} & \rightsquigarrow & [e_1]_{(\lambda n_1\ \ x_1.[e_2]_{(\lambda n_2\ \ x_2.x_1\ c\ n_2\ \ x_2),n_1}\ ),(n+1)}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \leadsto & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \leadsto & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if } n = l \textbf{ then } [\![c, n, \lambda c\ n\ l.e]\!] \textbf{ else } e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{{\color{red}c},n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![{\color{red}c},n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \leadsto & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{c,{\color{red}n},l} & \leadsto & \textbf{if}\ n = l\ \textbf{then}\ [\![c, {\color{red}n}, \lambda c\ n\ l.e]\!]\ \textbf{else}\ e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \leadsto & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\color{red}{\lambda c\ n\ l.e}]\!]\ \textbf{else}\ e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ x \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ {\color{red}e}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l}\\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l}\\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l}\\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e\\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),{\color{red}0},\infty}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ x\rangle_{c,n,l}\\
[\lambda x.e]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l}\\
[e_1\ e_2]_{c,n,l} & \leadsto & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l}\\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e\\
\text{PRIMOPS}\ e & \leadsto & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if } n=l \textbf{ then } [\![c,n,\lambda c\ n\ l.e]\!] \textbf{ else } e \\
\text{PRIMOPS } e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\color{red}{\text{CHECKPOINT}}\ e\ n & \rightsquigarrow & [e]_{\perp,0,n}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l}\\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l}\\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l}\\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e\\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}\\
\text{CHECKPOINT}\ e\ n & \rightsquigarrow & [e]_{\perp,{\color{red}0},n}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \leadsto & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\text{CHECKPOINT}\ e\ {\color{red}n} & \leadsto & [e]_{\perp,0,{\color{red}n}}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\text{CHECKPOINT}\ e\ n & \rightsquigarrow & [e]_{\perp,0,n}
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if } n=l \textbf{ then } [\![c,n,\lambda c\ n\ l.e]\!] \textbf{ else } e \\
\text{PRIMOPS } e & \leadsto & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\text{CHECKPOINT } e\ n & \leadsto & [e]_{\perp,0,n} \\
\text{RESUME } [\![c,n,v]\!] & = & v\ c\ n\ \infty
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if}\ n=l\ \textbf{then}\ [\![c,n,\lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\text{CHECKPOINT}\ e\ n & \rightsquigarrow & [e]_{\bot,0,n} \\
\text{RESUME}\ [\![c,n,v]\!] & = & v\ c\ n\ \infty
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ x\rangle_{c,n,l} \\
[\lambda x.e]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l} \\
[e_1\ e_2]_{c,n,l} & \leadsto & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l} \\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if}\ n=l\ \textbf{then}\ [\![c, n, \lambda c\ n\ l.e]\!]\ \textbf{else}\ e \\
\text{PRIMOPS}\ e & \leadsto & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty} \\
\text{CHECKPOINT}\ e\ n & \leadsto & [e]_{\perp,0,n} \\
\text{RESUME}\ [\![c, n, v]\!] & = & v\ c\ n\ \infty
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ x\rangle_{c,n,l}\\
[\lambda x.e]_{c,n,l} & \leadsto & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l}\\
[e_1\ e_2]_{c,n,l} & \leadsto & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l}\\
\langle e\rangle_{c,n,l} & \leadsto & \textbf{if } n=l \textbf{ then } [\![c,n,\lambda c\ n\ l.e]\!] \textbf{ else } e\\
\text{PRIMOPS } e & \leadsto & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}\\
\text{CHECKPOINT } e\ n & \leadsto & [e]_{\bot,0,n}\\
\text{RESUME } [\![c,n,v]\!] & = & v\ c\ n\ \infty
\end{array}
$$

CPS Conversion to Support the General Checkpointing API

$$
\begin{array}{lcl}
[x]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ x\rangle_{c,n,l}\\
[\lambda x.e]_{c,n,l} & \rightsquigarrow & \langle c\ (n+1)\ l\ \lambda c\ n\ l\ x.[e]_{c,n,l}\rangle_{c,n,l}\\
[e_1\ e_2]_{c,n,l} & \rightsquigarrow & \langle [e_1]_{(\lambda n_1\ l_1\ x_1.[e_2]_{(\lambda n_2\ l_2\ x_2.x_1\ c\ n_2\ l_2\ x_2),n_1,l_1}),(n+1),l}\rangle_{c,n,l}\\
\langle e\rangle_{c,n,l} & \rightsquigarrow & \textbf{if } n=l \textbf{ then } [\![c,n,\lambda c\ n\ l.e]\!] \textbf{ else } e\\
\text{PRIMOPS } e & \rightsquigarrow & [e]_{(\lambda n\ l\ x.(x,n)),0,\infty}\\
\text{CHECKPOINT } e\ n & \rightsquigarrow & [e]_{\bot,0,n}\\
\text{RESUME } [\![c,n,v]\!] & = & v\ c\ n\ \infty
\end{array}
$$

Space and Time Complexity

computation length	t
maximal live storage	w
space for checkpoints	$O(w \log t)$
space for tape	$O(w)$
time to (re)compute primal	$O(t \log t)$
time for reverse sweep	$O(t)$

FORTRAN Example

```
      subroutine f(x, y)
      n = 100003
      y = x
c$ad binomial-ckp n+1 30 1
      do i = 1, n
         m = l(x, i)
         do j = 1, m
            y = y*y
            y = sqrt(y)
         end do
      end do
      end
```

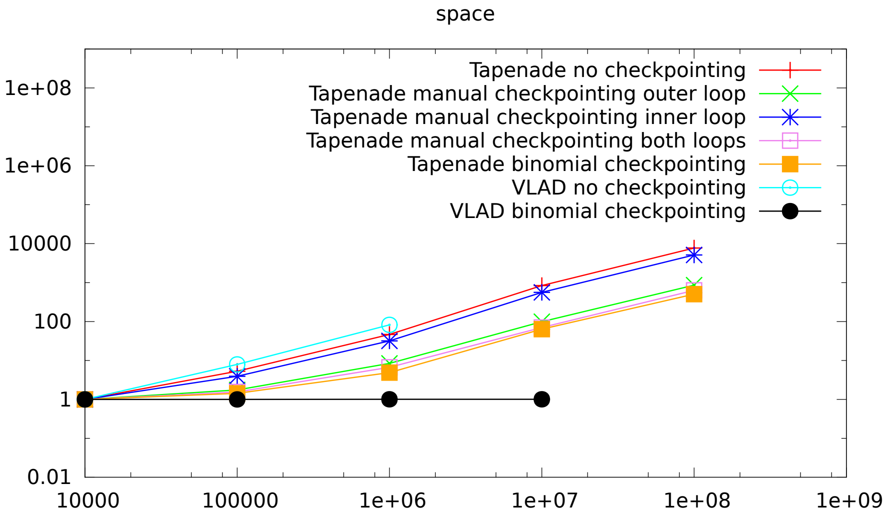
VLAD Example

```
(define (f x)
 (let ((n 100003))
  (let outer ((i 1) (y x))
   (if (> i n)
       y
       (outer (+ i 1)
              (let ((m (l x i)))
               (let inner ((j 1) (y y))
                (if (> j m)
                    y
                    (inner (+ j 1)
                           (sqrt (* y y))))))))))))
```

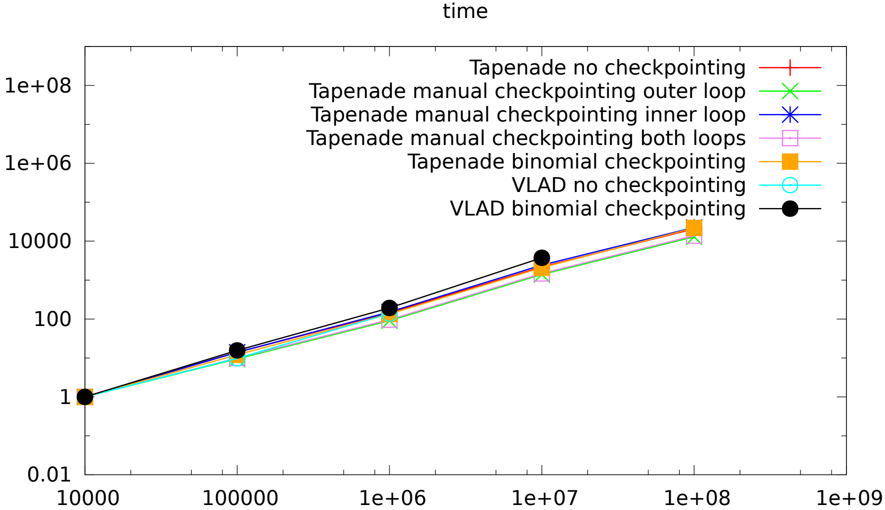
Space and Time Complexity of our Example

computation length	t	$= O(n)$
maximal live storage	w	$= O(1)$
space for checkpoints	$O(w \log t)$	$= O(\log n)$
space for tape	$O(w)$	$= O(1)$
time to (re)compute primal	$O(t \log t)$	$= O(n \log n)$
time for reverse sweep	$O(t)$	$= O(n)$

Space Usage of our Example



Time Usage of our Example



Take-Home Message

Checkpointing

- is traditionally formulated around loop iterations
- but can be extended to arbitrary code
- that doesn't have same-size iterations of a single loop
- using CPS to make arbitrary code look like it does

metaphor: a CPU is an instruction-execution loop