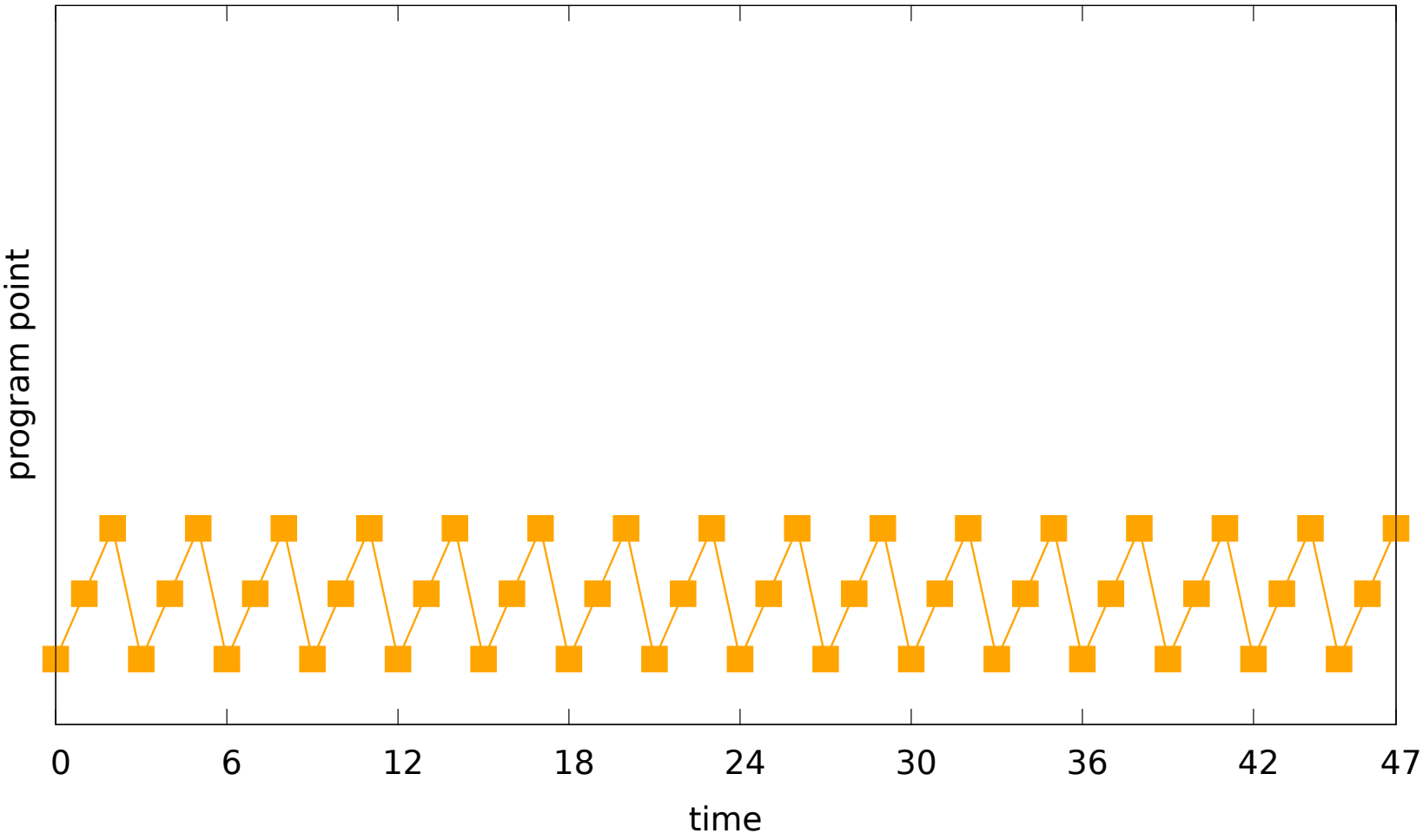
Binomial Checkpointing for Arbitrary Programs with No User Annotation

Jeffrey Mark Siskind, qobi@purdue.edu
Barak Avrum Pearlmutter, barak@pearlmutter.net

PURDUE UNIVERSITY

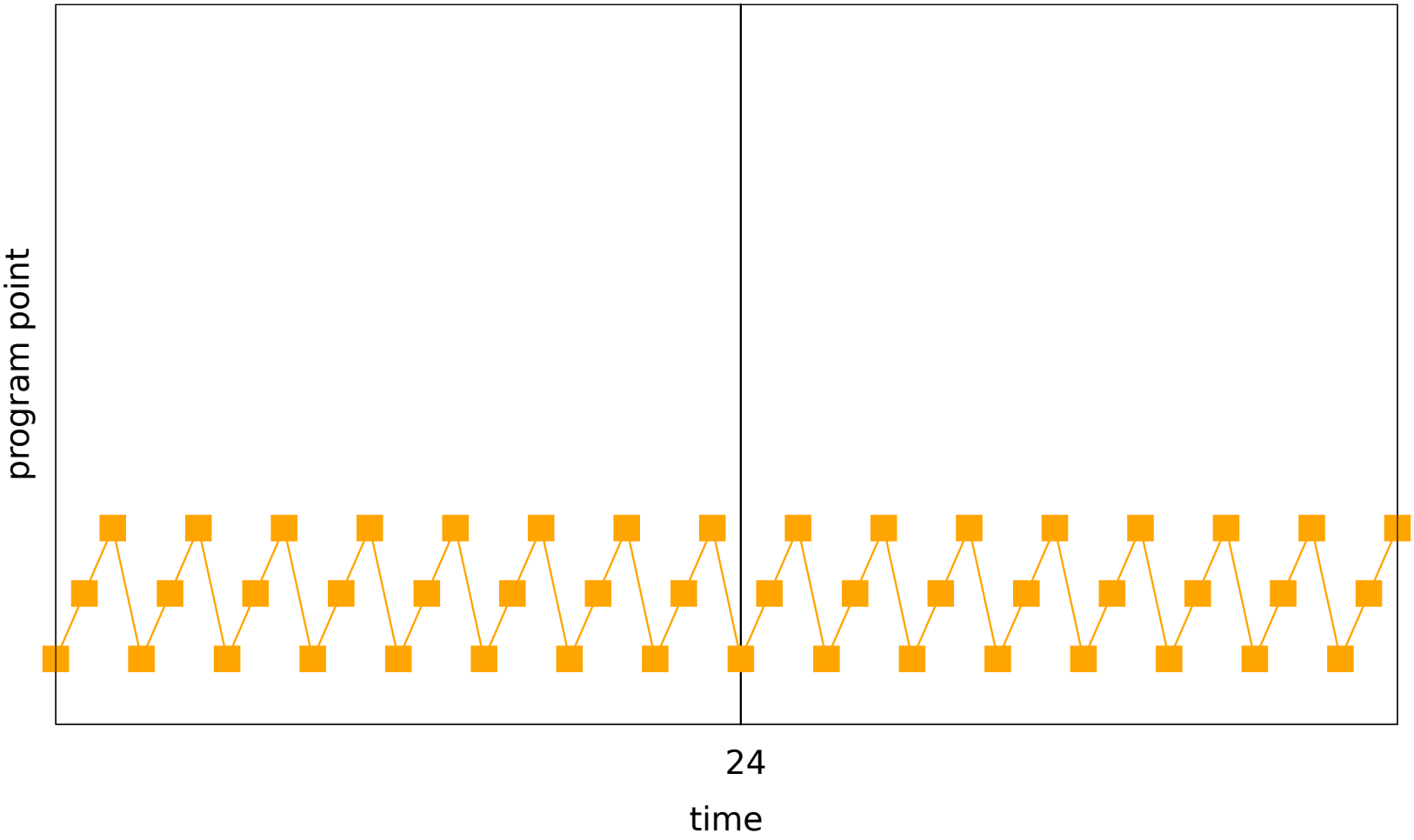
AD2016, Wednesday 14 September 2016

Execution Trace of Loop



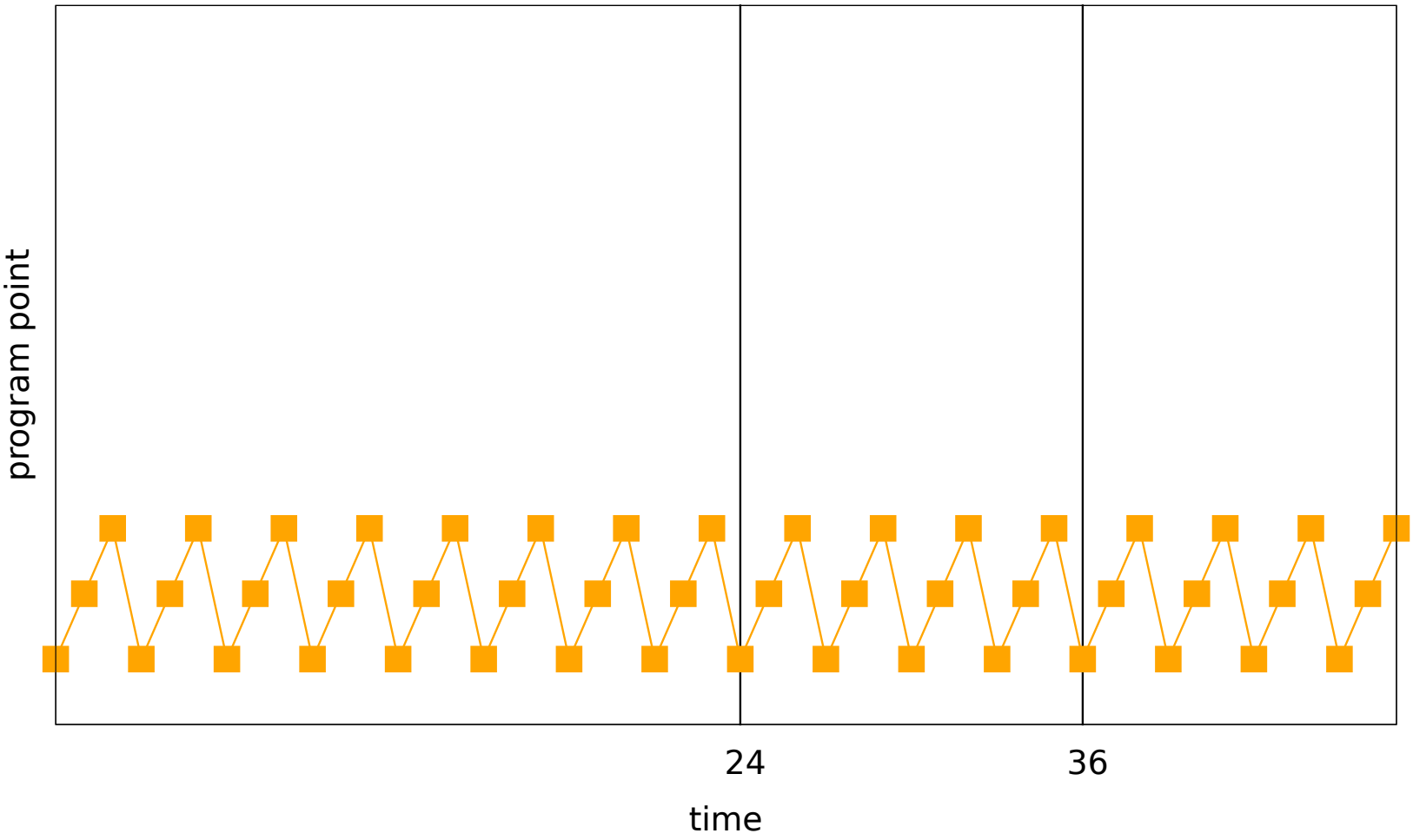
Execution Trace of Loop

Easy to make regular and uniform checkpoints



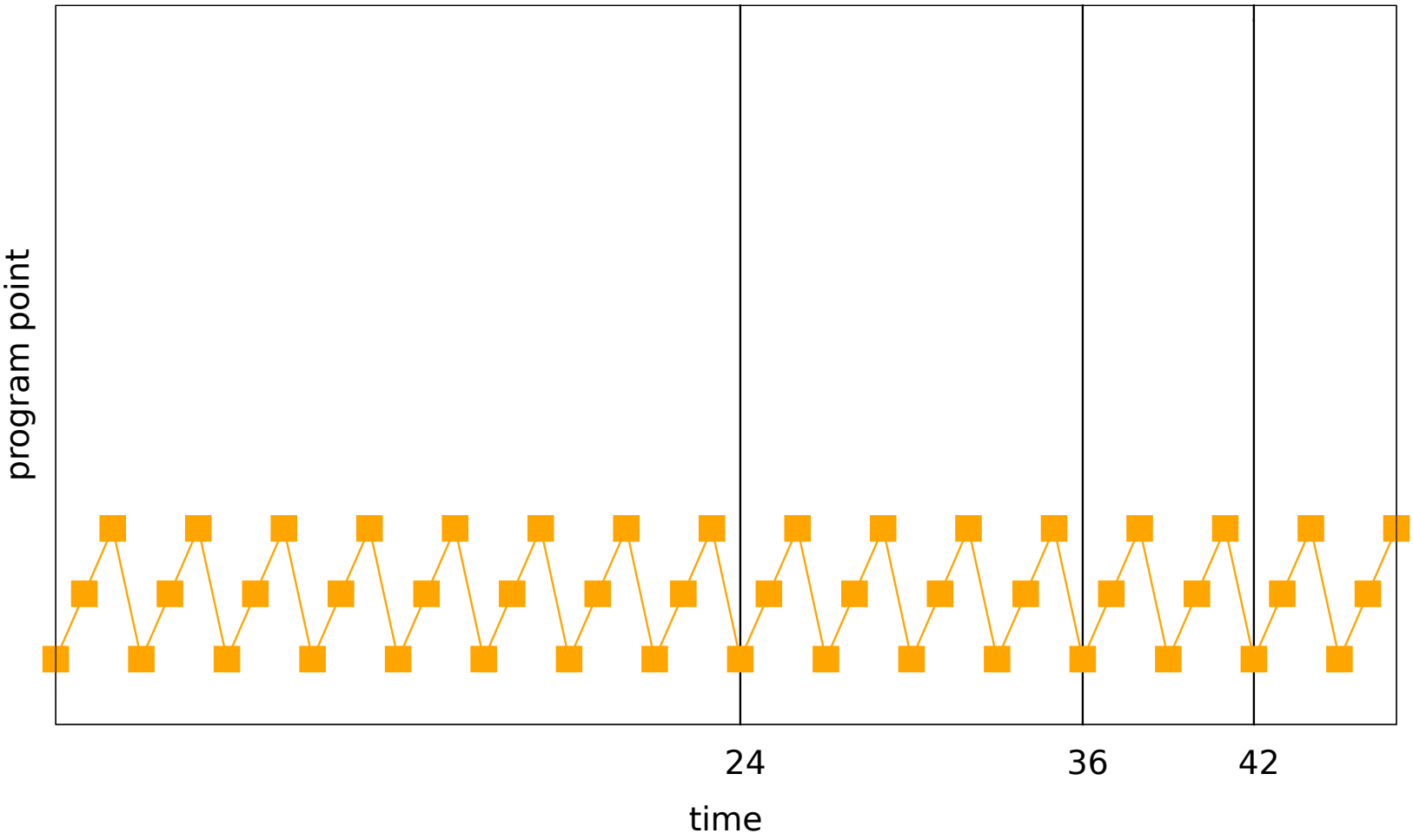
Execution Trace of Loop

Easy to make regular and uniform checkpoints



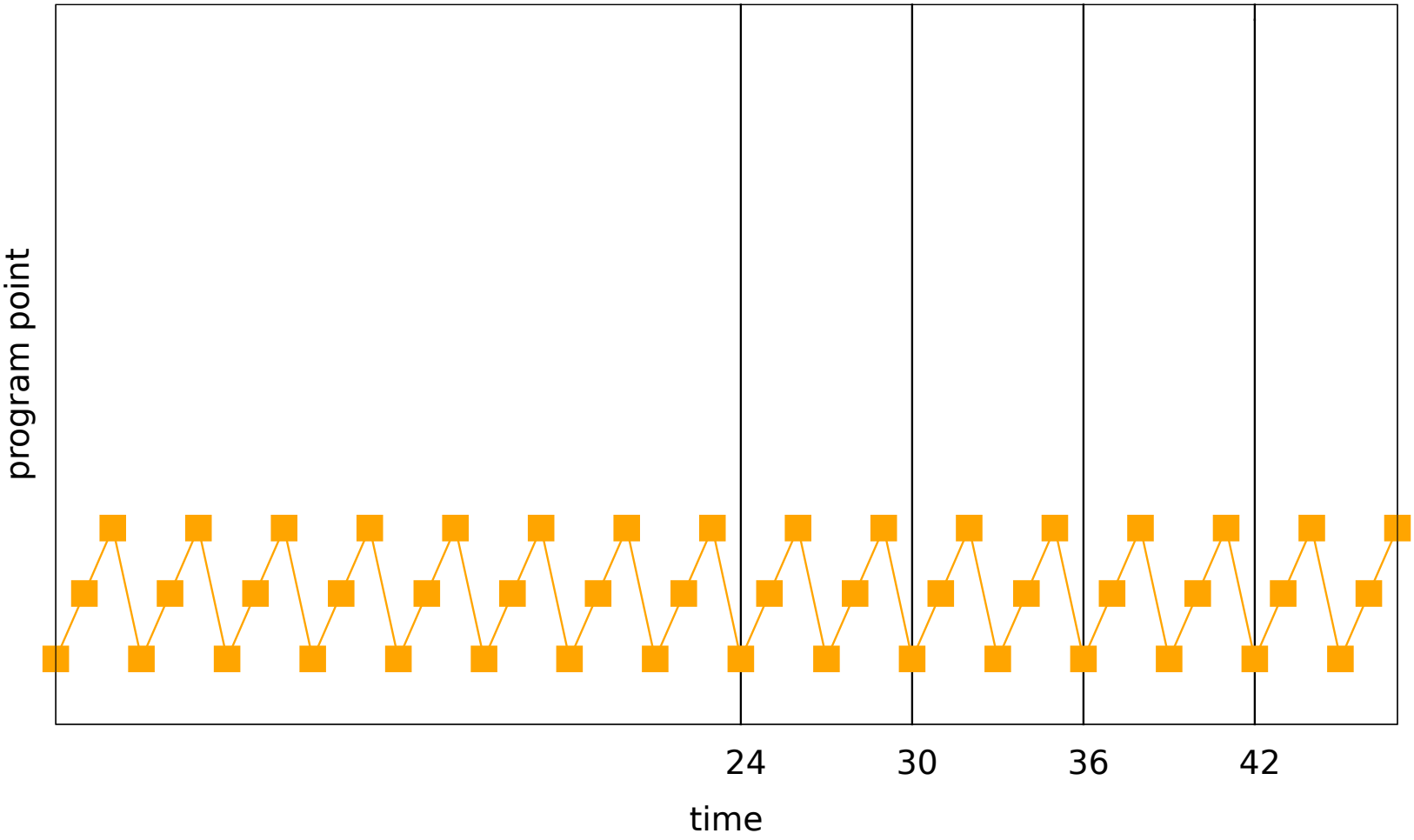
Execution Trace of Loop

Easy to make regular and uniform checkpoints



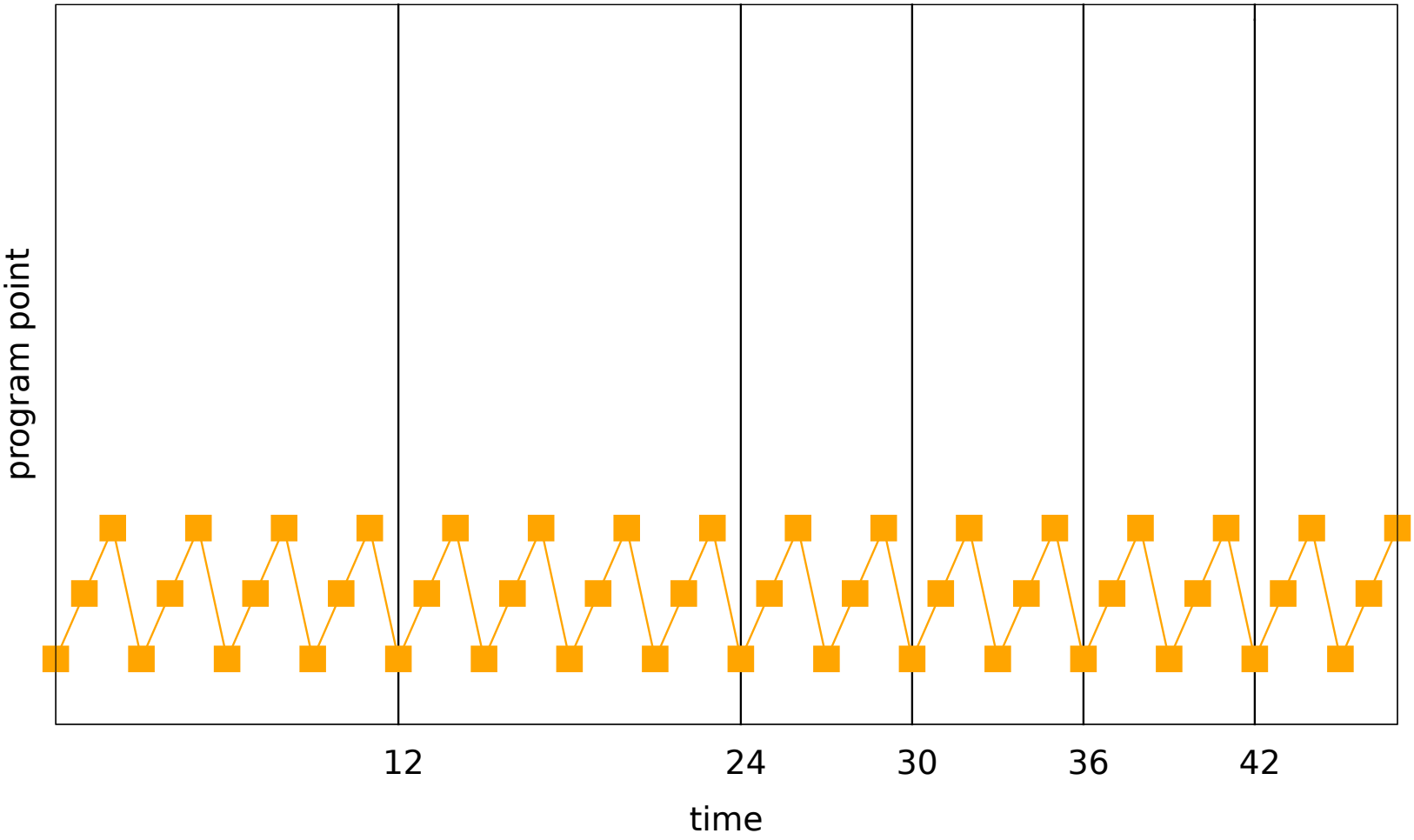
Execution Trace of Loop

Easy to make regular and uniform checkpoints



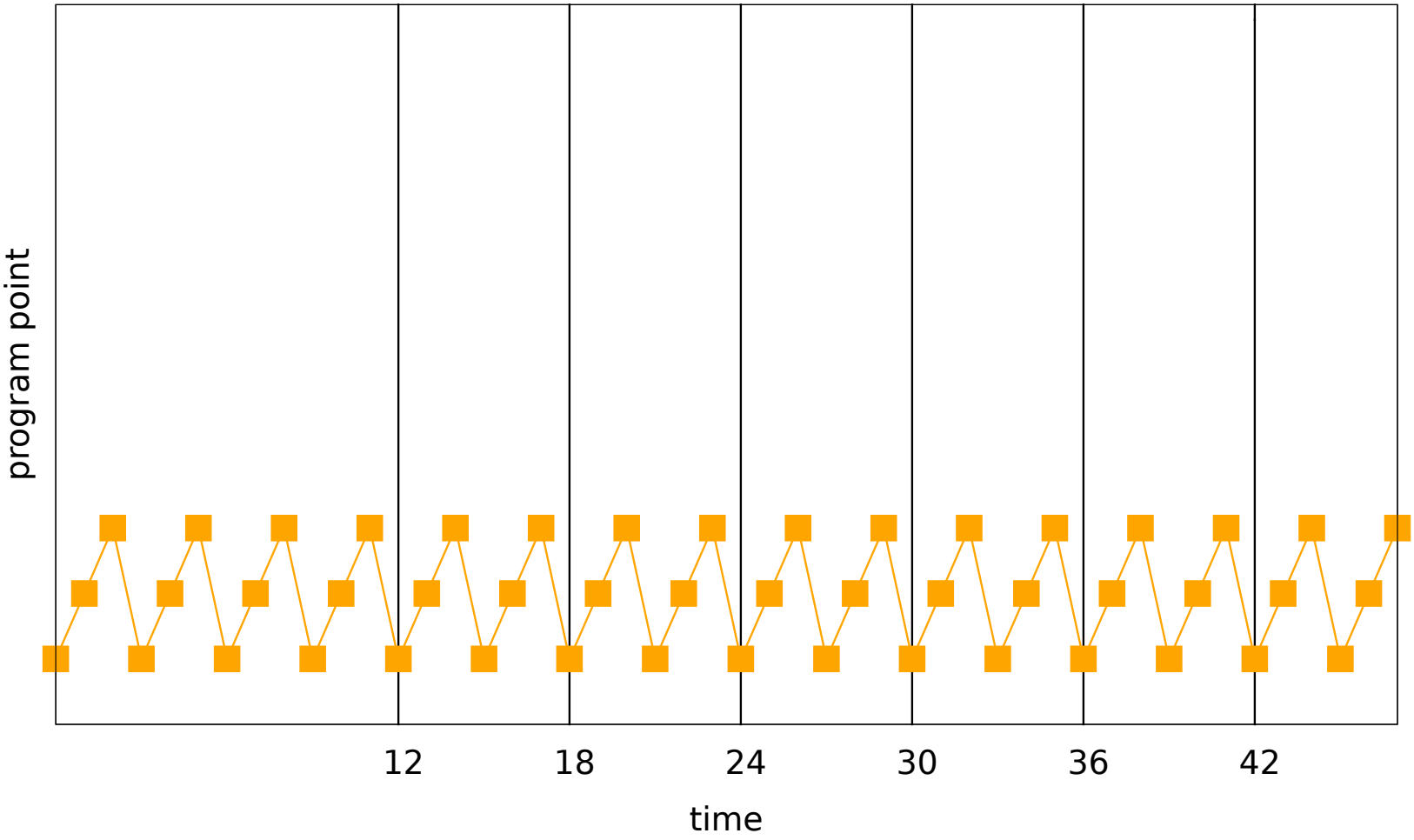
Execution Trace of Loop

Easy to make regular and uniform checkpoints



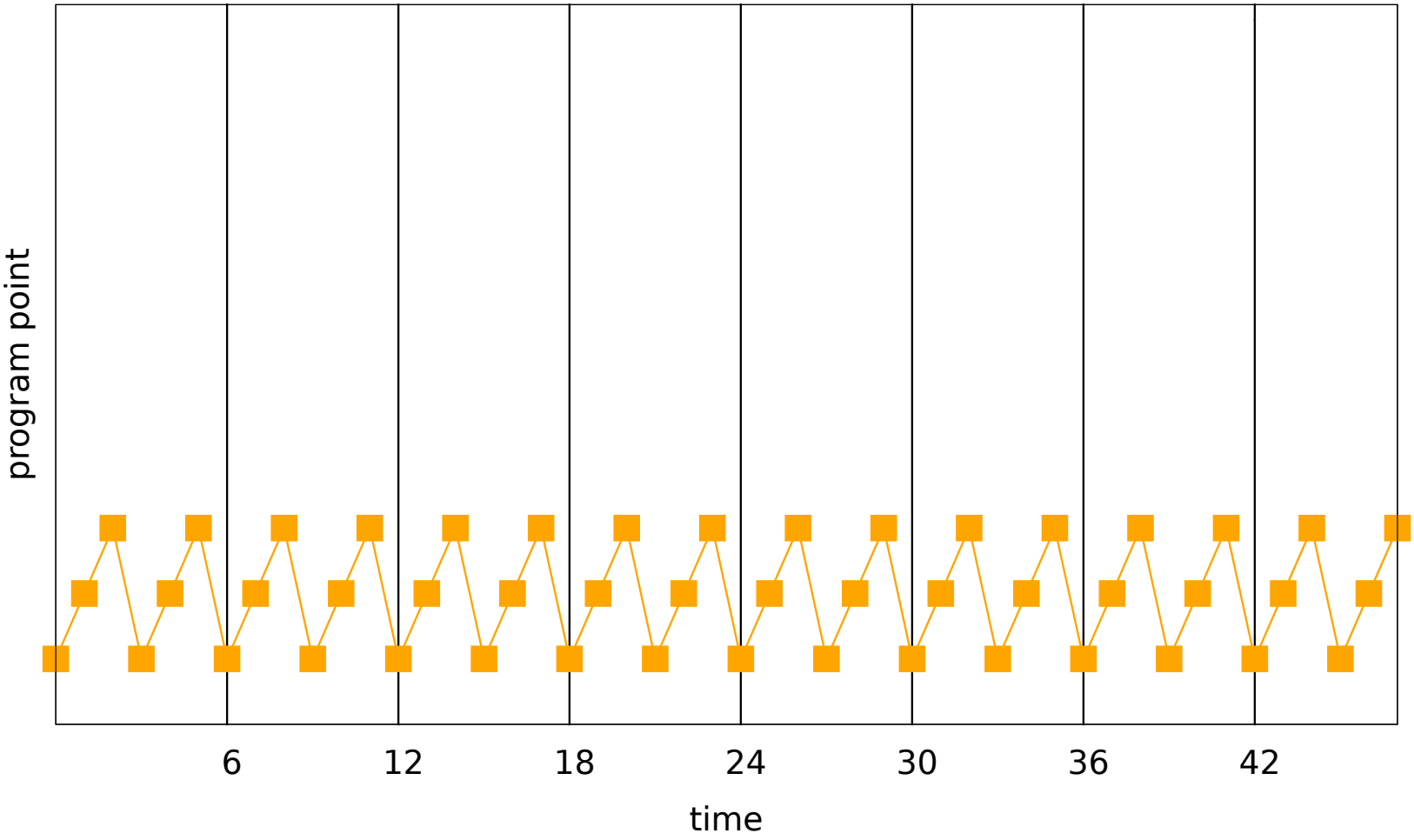
Execution Trace of Loop

Easy to make regular and uniform checkpoints

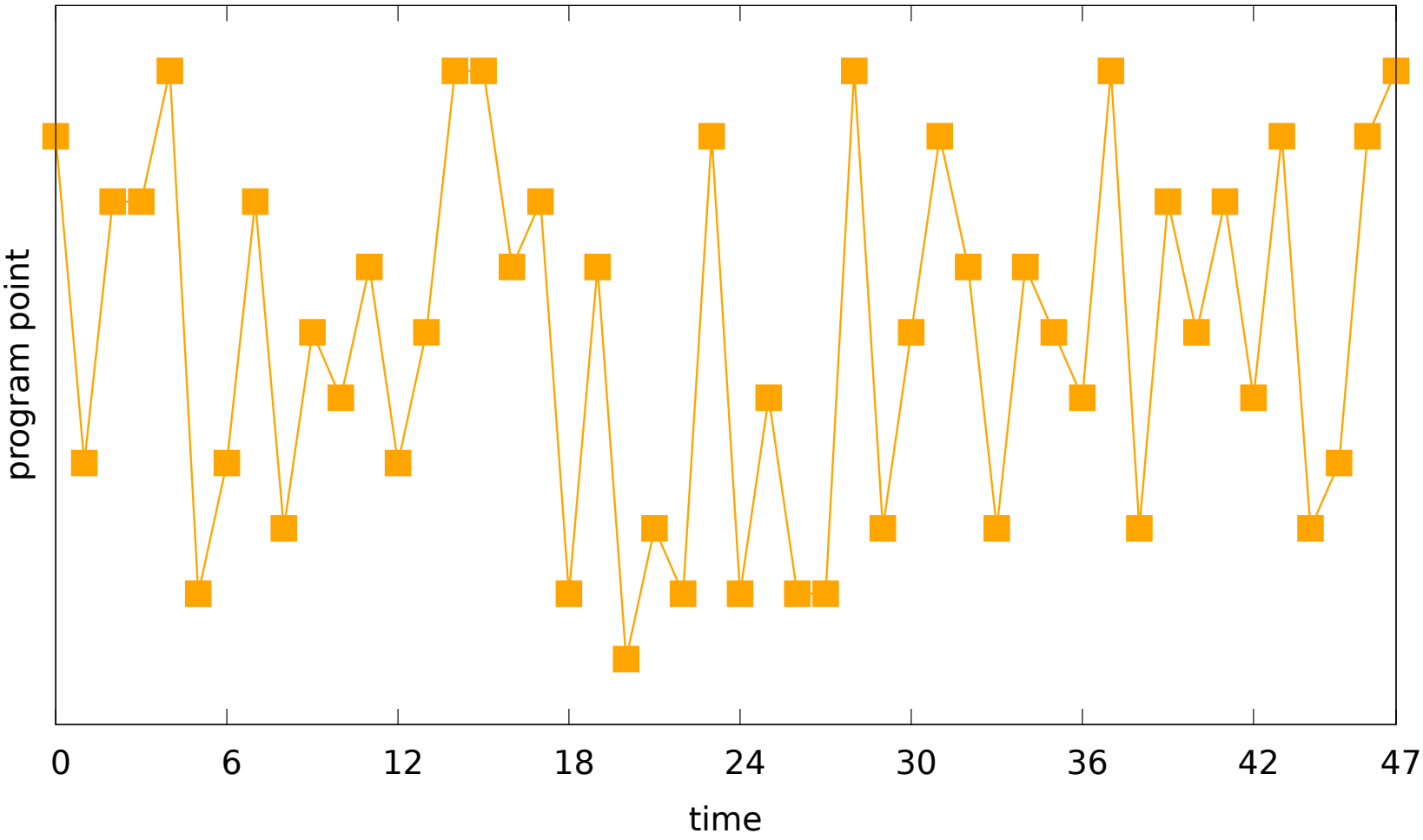


Execution Trace of Loop

Easy to make regular and uniform checkpoints

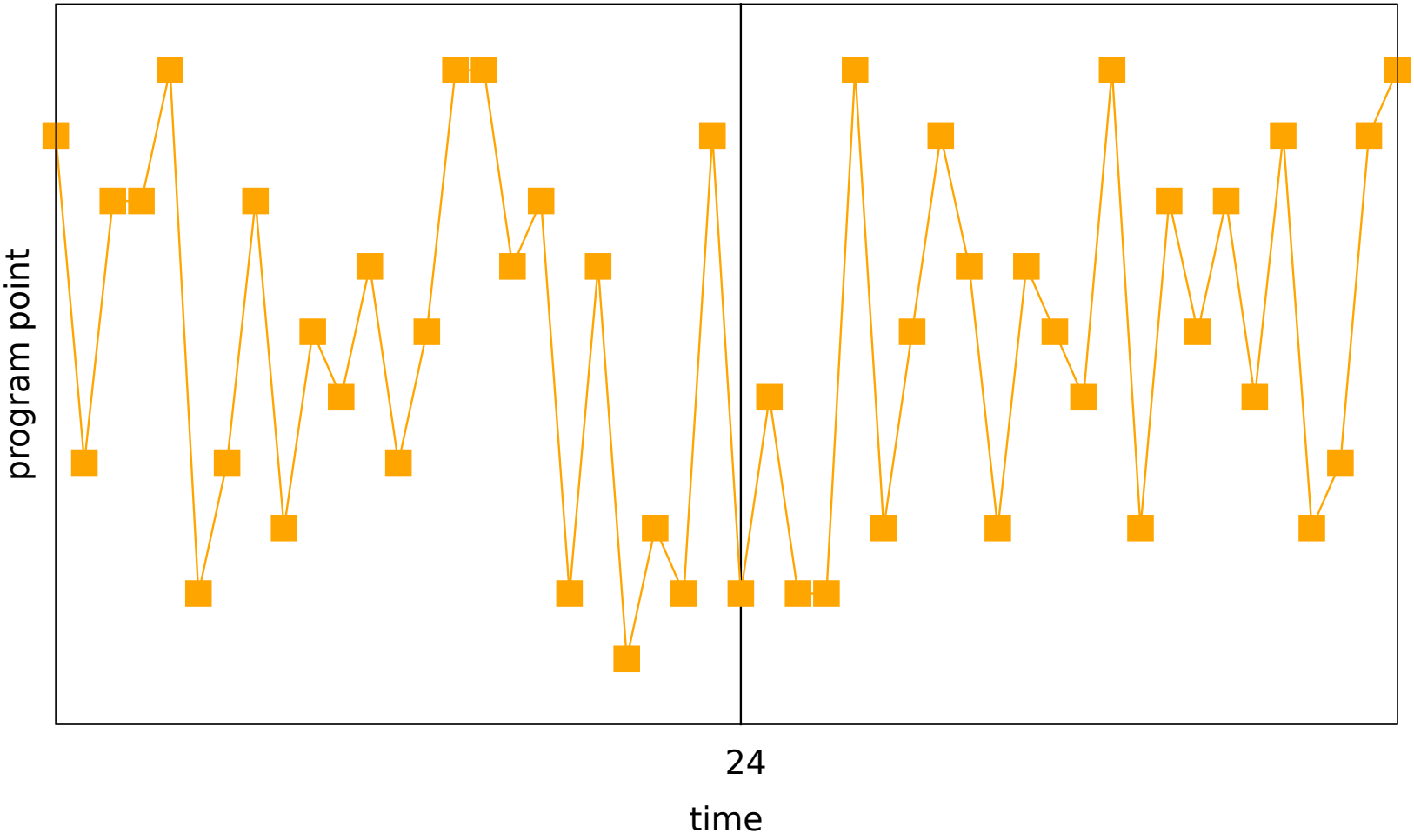


Execution Trace of Arbitrary Code



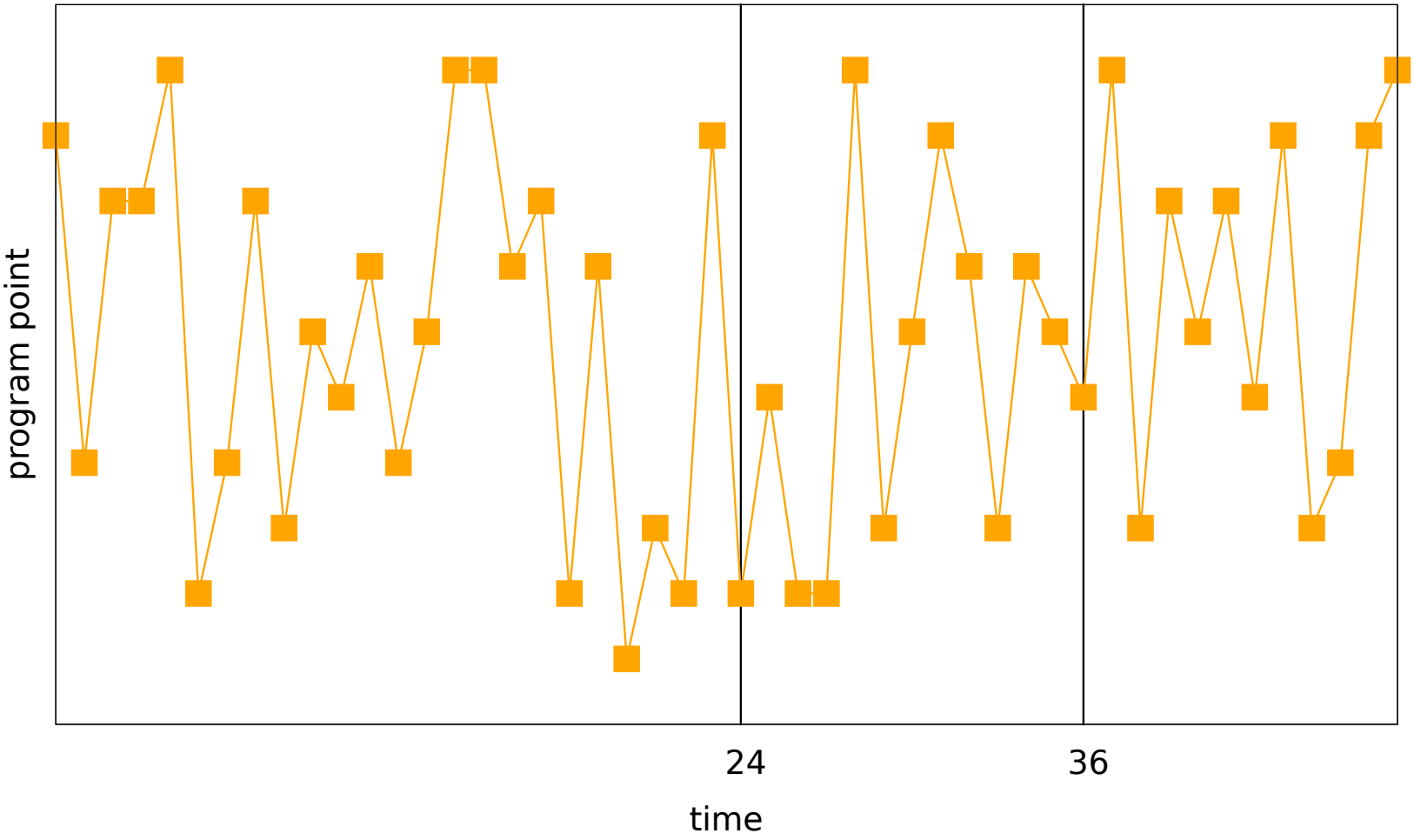
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



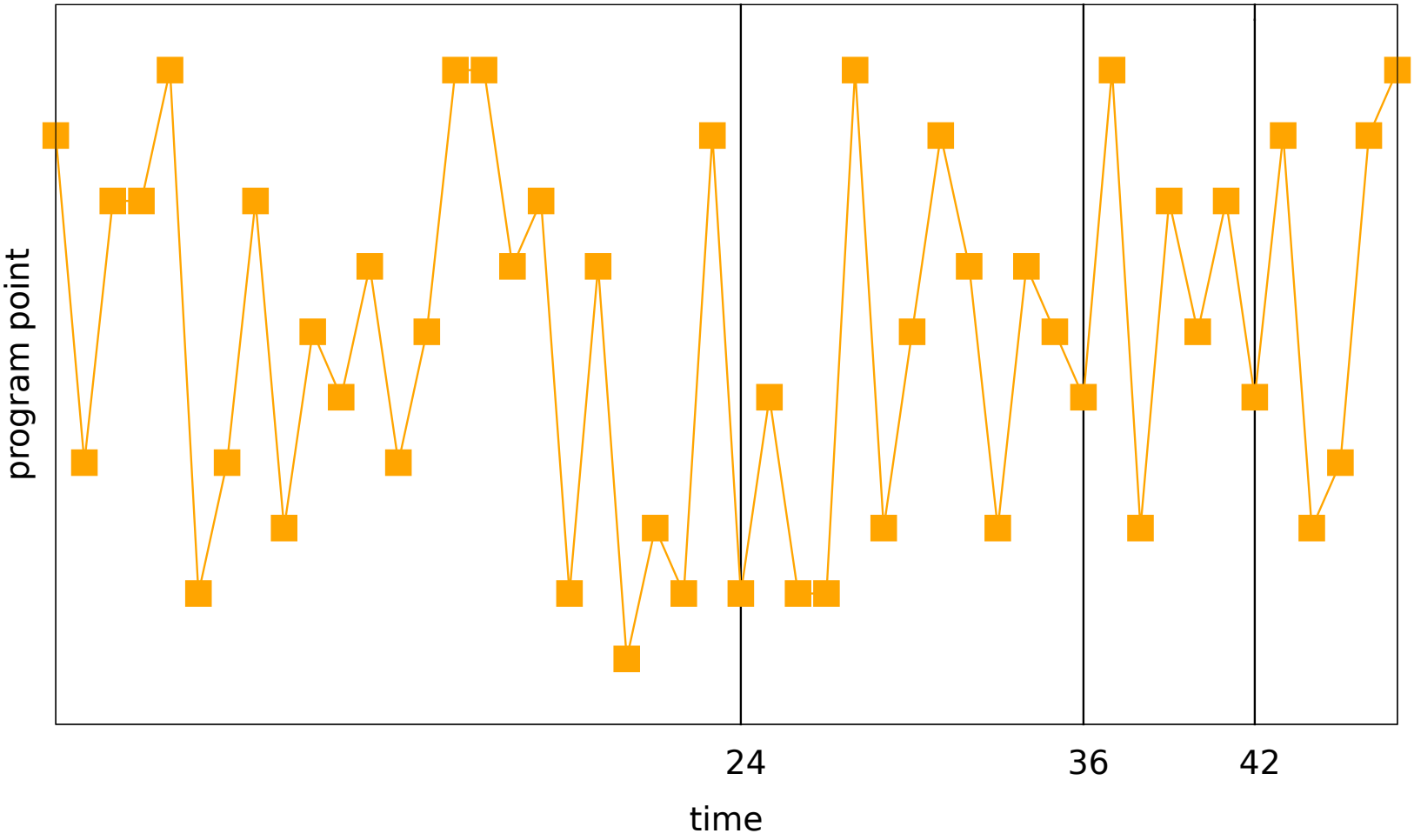
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



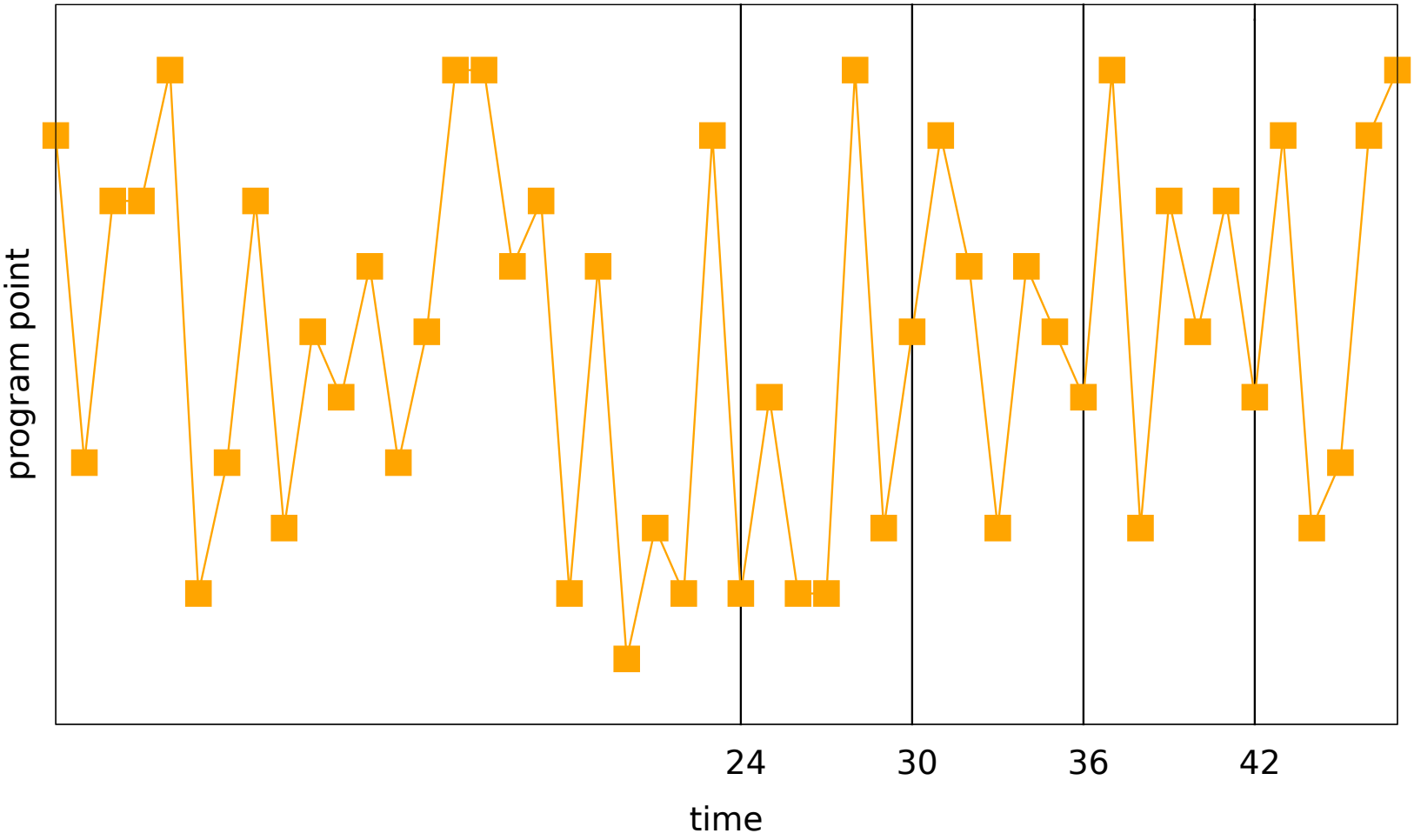
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



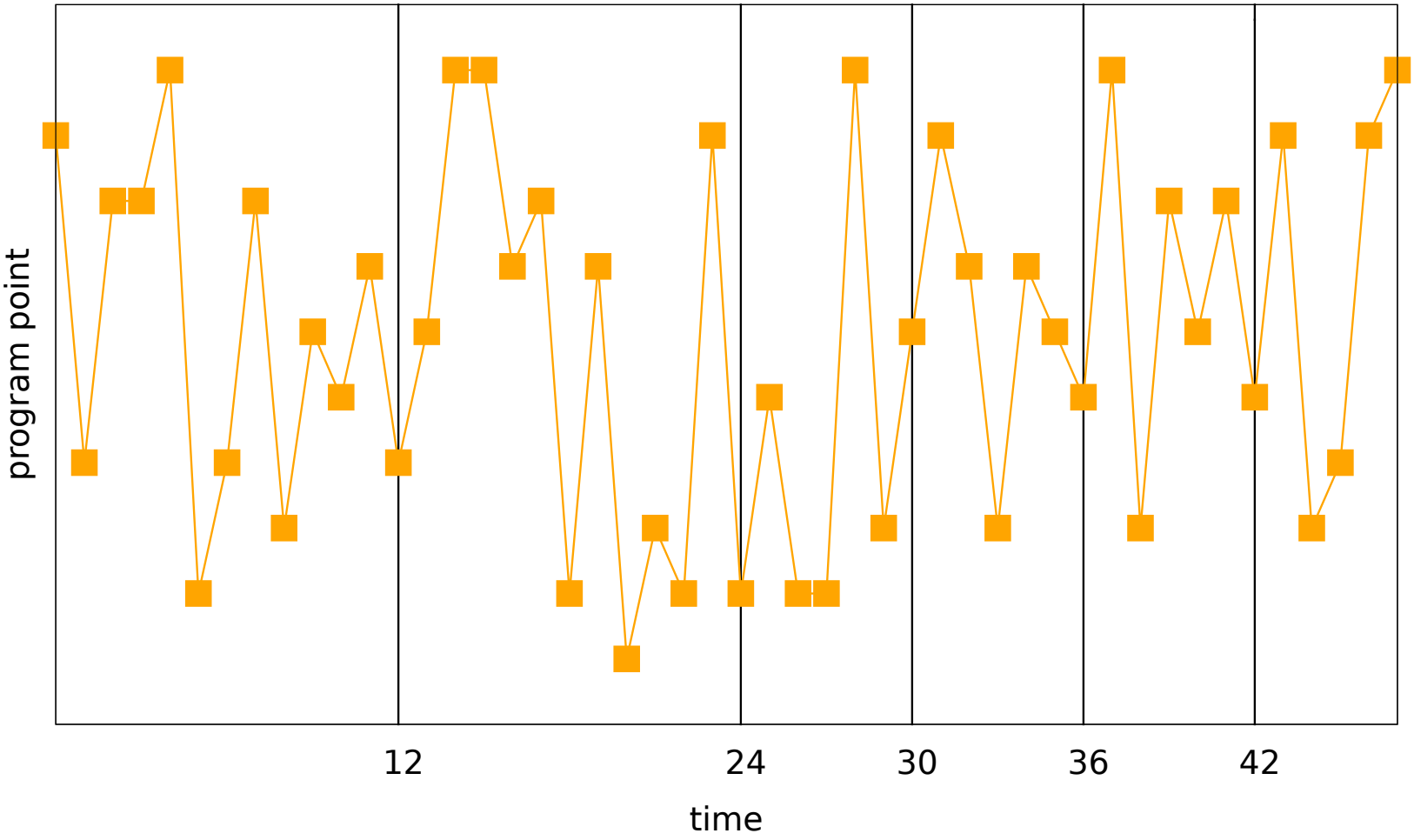
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



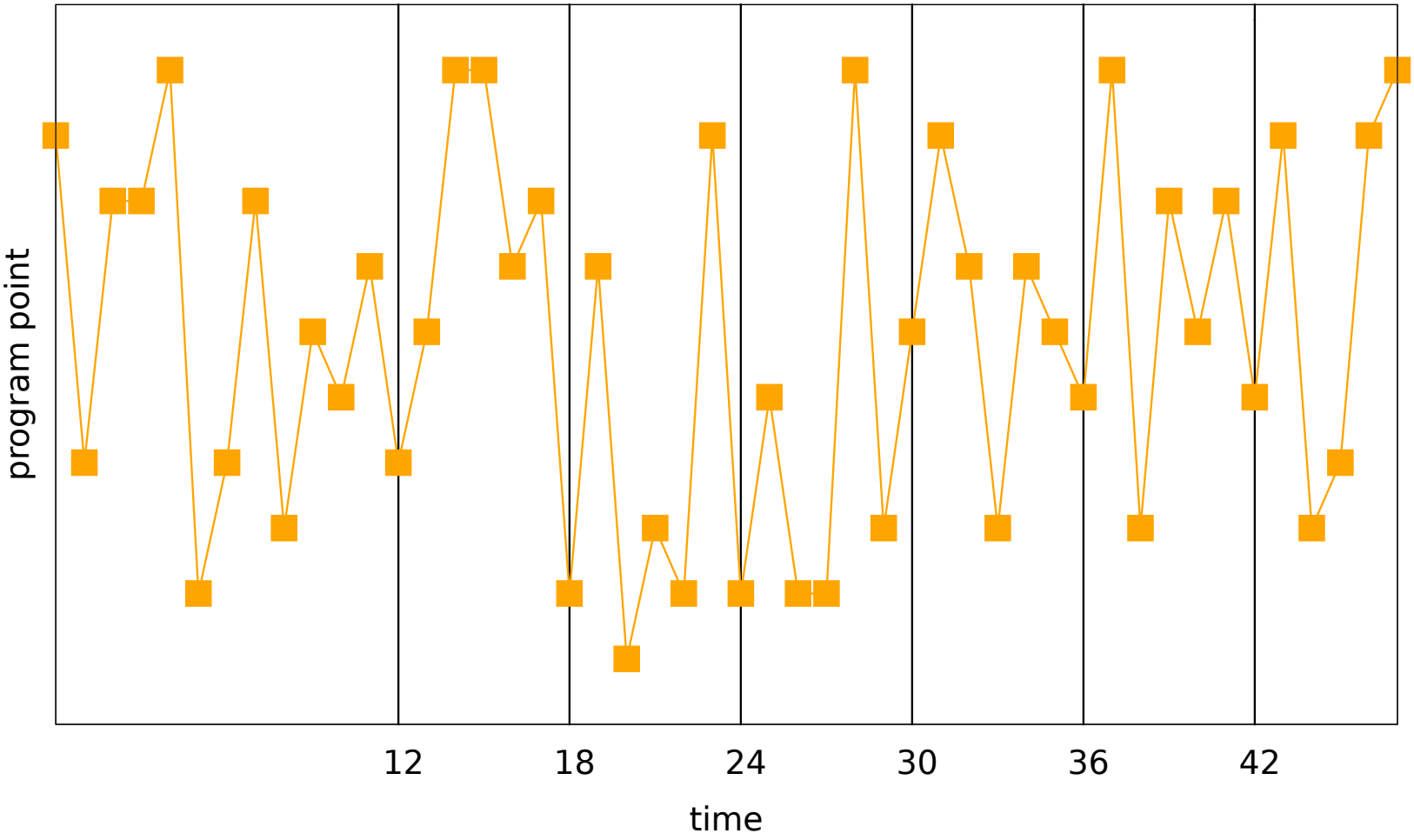
Execution Trace of Arbitrary Code

Difficult to make regular and uniform checkpoints



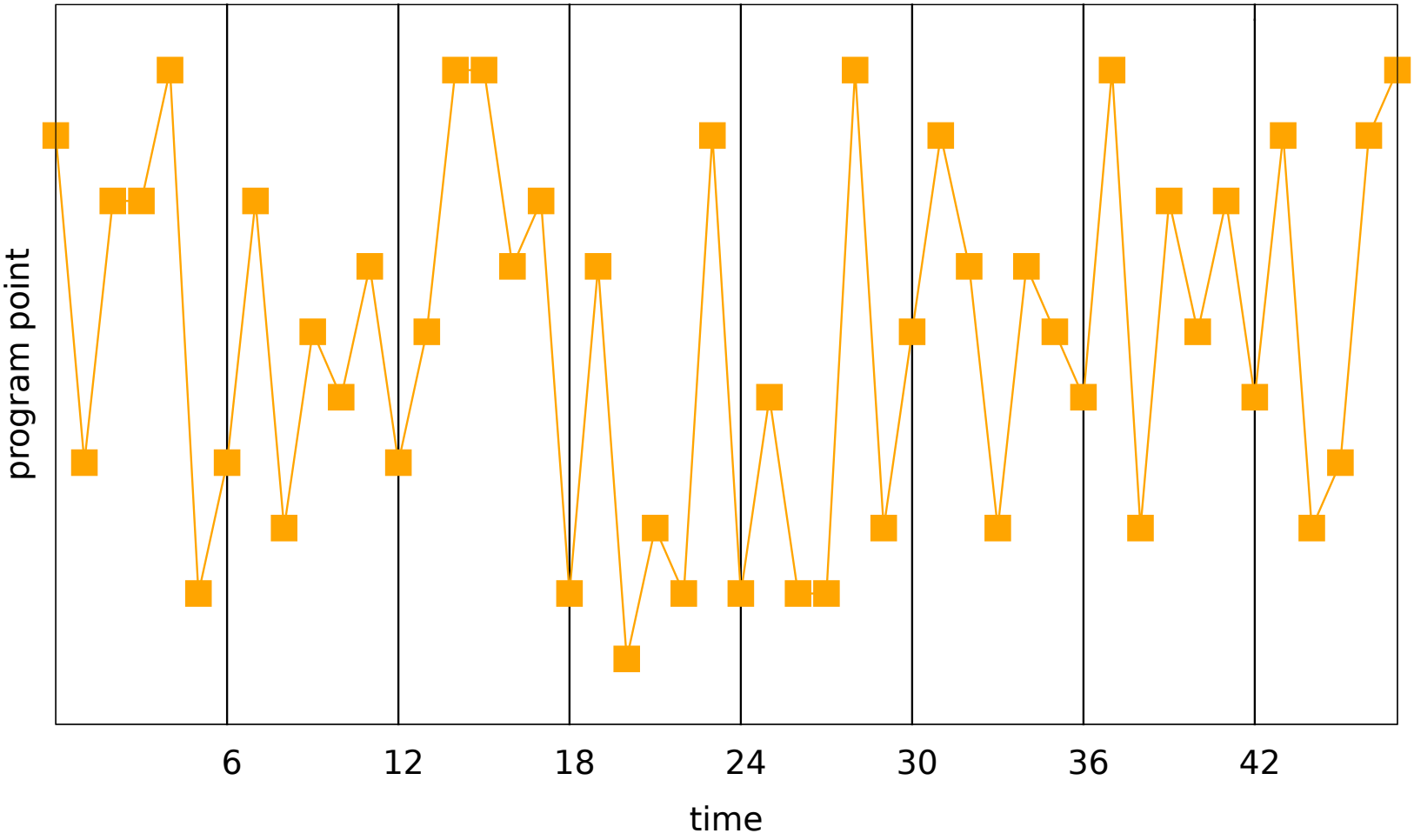
Execution Trace of Arbitrary Code

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
```

$\rightsquigarrow$

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case: $h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case: $h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case:

$h \circ g = f$ (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)

Algorithm for Bisection Checkpointing

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

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

inductive case: $h \circ g = f$ (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)

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

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

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

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 $y = f(x)$.

Bisection Checkpointing via General Checkpointing API

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

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

inductive case:

$h \circ g = f$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

inductive case: $h \circ g = f$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

inductive case:

$h \circ g = f$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

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)

Bisection Checkpointing via General Checkpointing API

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

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

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)

Bisection Checkpointing via General Checkpointing API

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

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

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)

Bisection Checkpointing via General Checkpointing API

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

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

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

$u = \text{CHECKPOINT}\ f\ x\ n$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

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

$u = \text{CHECKPOINT}\ f\ x\ n$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

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

$u = \text{CHECKPOINT}\ f\ x\ n$ (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)

Bisection Checkpointing via General Checkpointing API

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

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

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)

$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u}$ (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): $(y, \grave{x}) = \overset{\leftarrow}{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

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)

$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\ g\ x\ \grave{u}$ (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): $(y, \grave{x}) = \overset{\leftarrow}{\mathcal{J}}\ f\ x\ \grave{y}$ (0)

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)

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

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

inductive case:

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

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

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

$$(u, \grave{x}) = \overset{\checkmark}{\mathcal{J}}\, (\lambda x.\text{CHECKPOINT}\ f\ x\ n)\ x\ \grave{u} \tag{4}$$

CPS Conversion

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

$\rightsquigarrow$

```
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

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

CPS Conversion is a Program Transformation

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

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

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

CPS Conversion is a Program Transformation

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

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

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

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

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

CPS Conversion to Support the General Checkpointing API

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

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
```

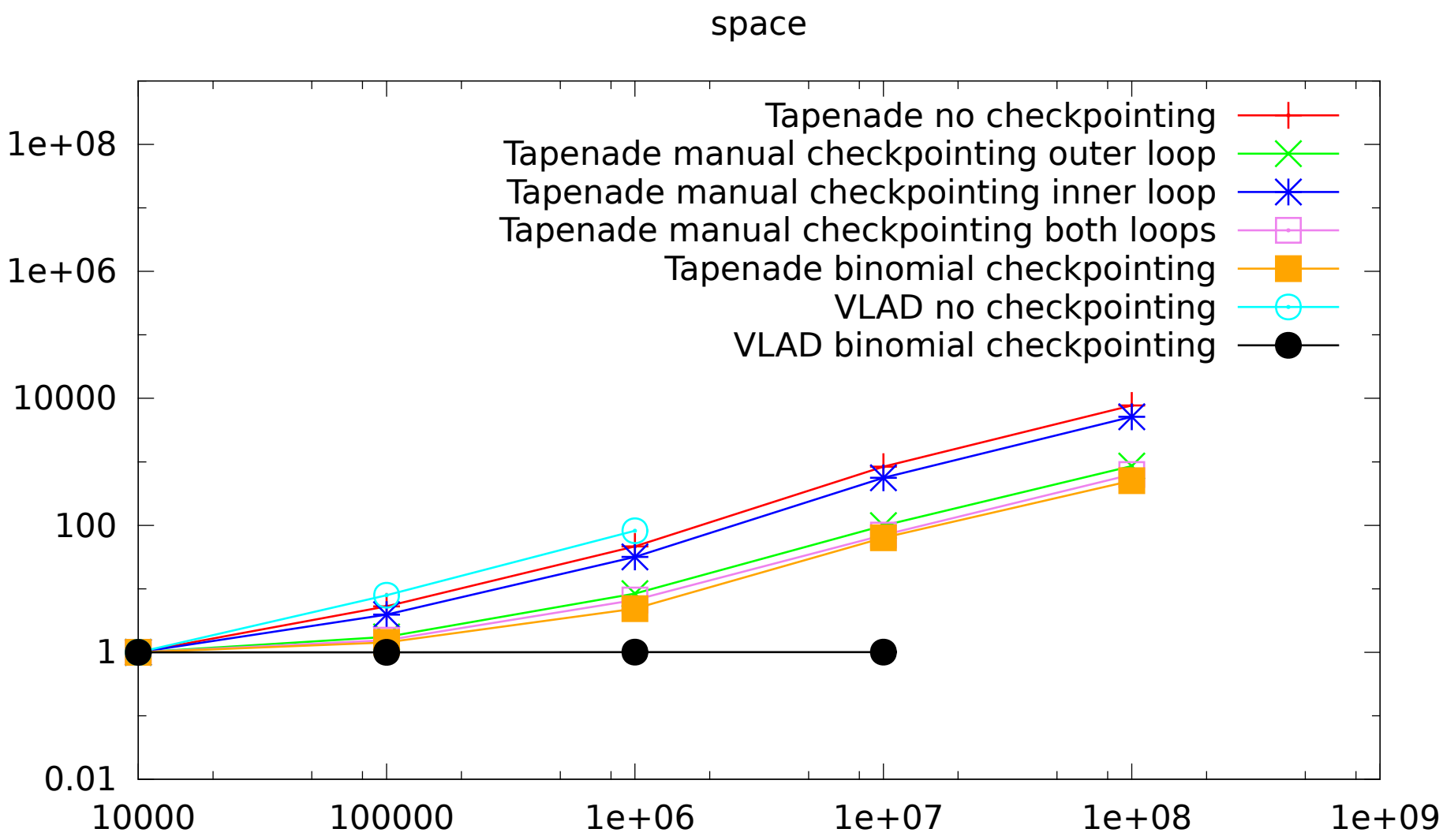
VLAD Example

```
(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)))))))))))
```

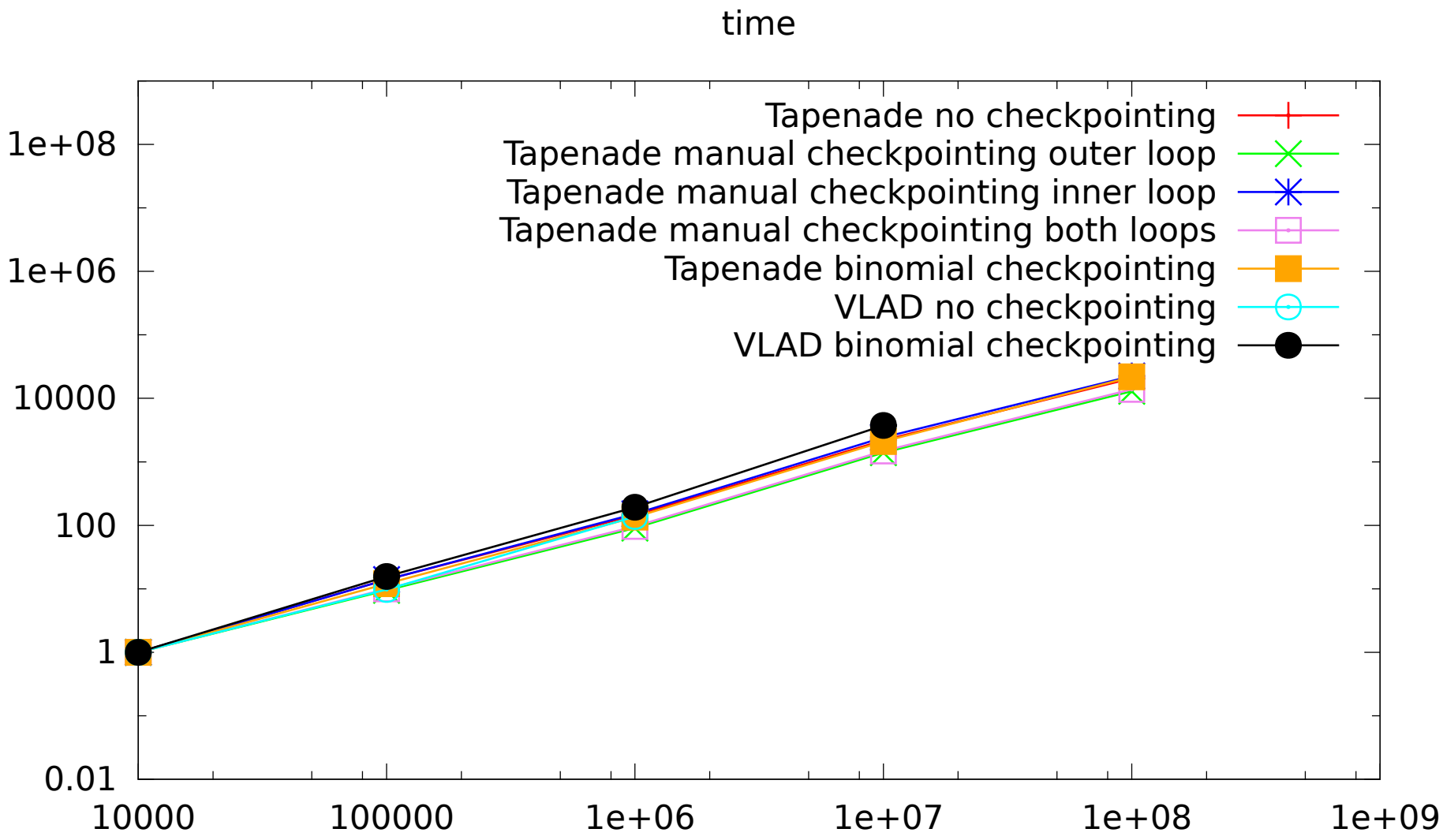
Space and Time Complexity of our Example

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