Vrije Universiteit Brussel

Software Languages.Lab

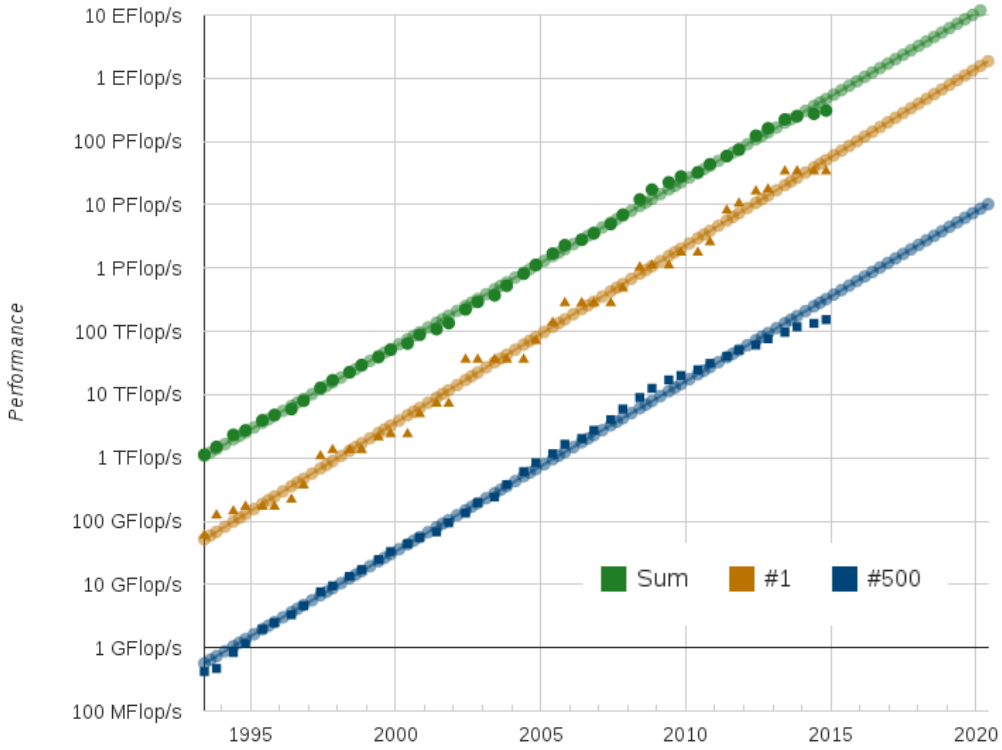
Resilient Distributed Concurrent Collections

Cédric Bassem

Promotor: Prof. Dr. Wolfgang De Meuter

Advisor: Dr. Yves Vandriessche

Evolution of Performance in High Performance Computing



(source: http://www.top500.org/statistics/perfdevel/)

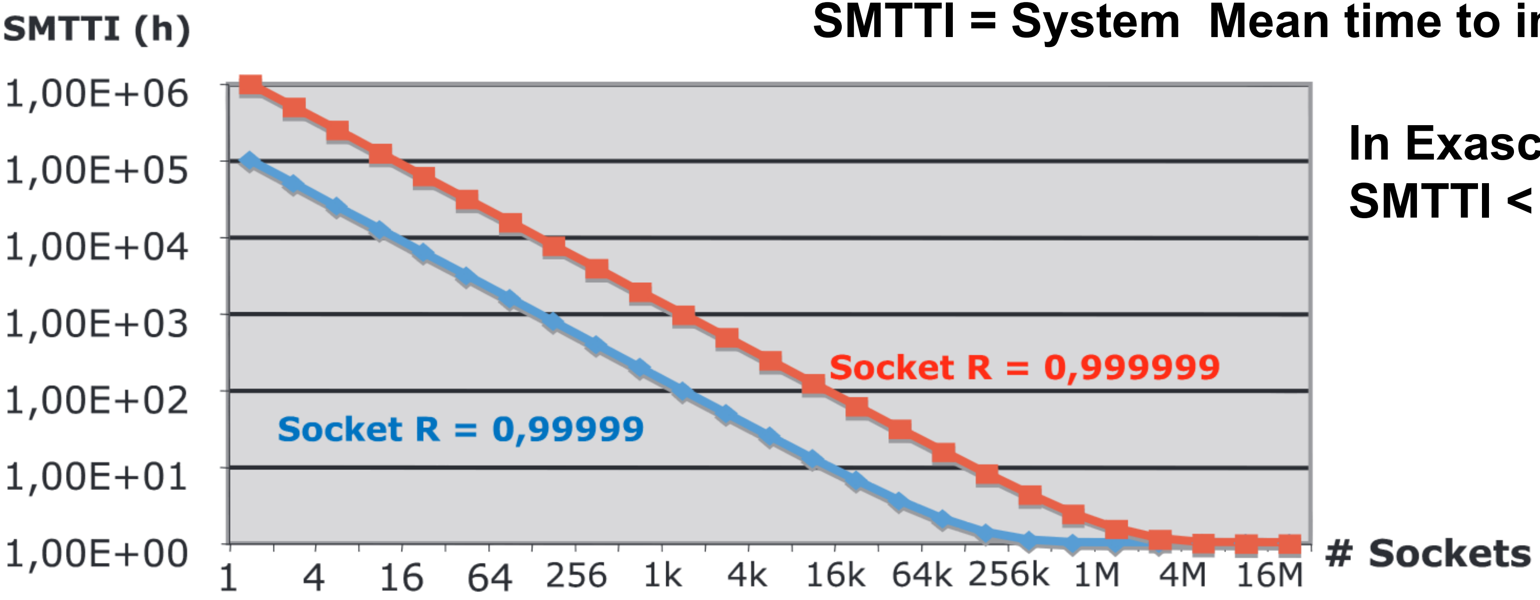
Exascale = 10^{18} Flop/s

Petascale = 10^{15} Flop/s

Evolution of Failures in HPC

Main Source: Hardware Faults (~ 50%)

SMTTI = System Mean time to interrupt



In Exascale
SMTTI < 30 min

Source: Franck Cappello (2009)

Resilience

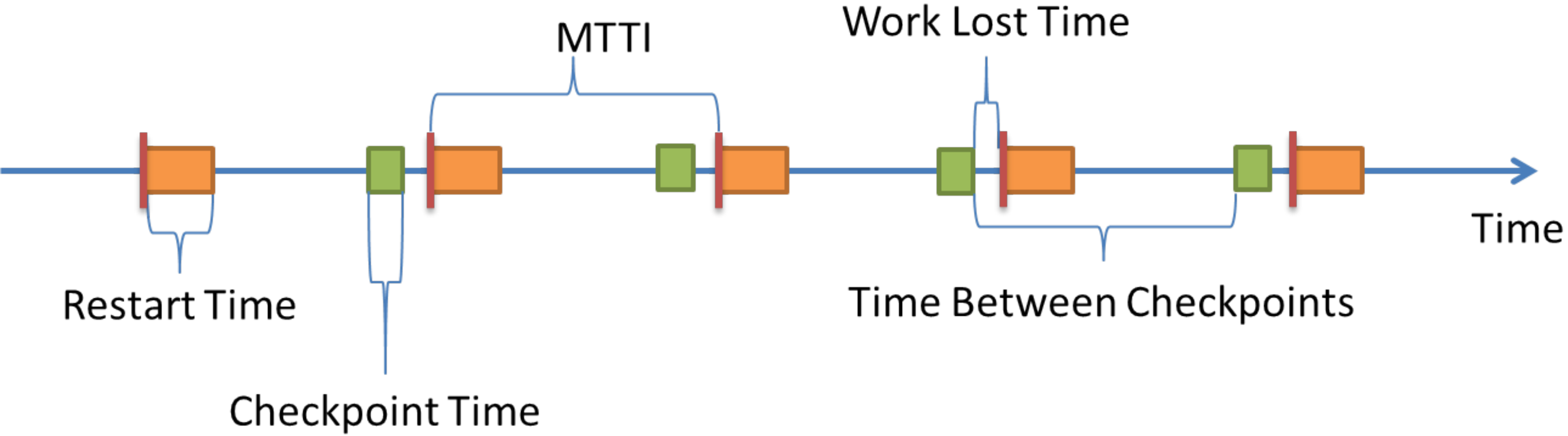
Resilience = Fault Tolerance

Avizienis et al. (2004)

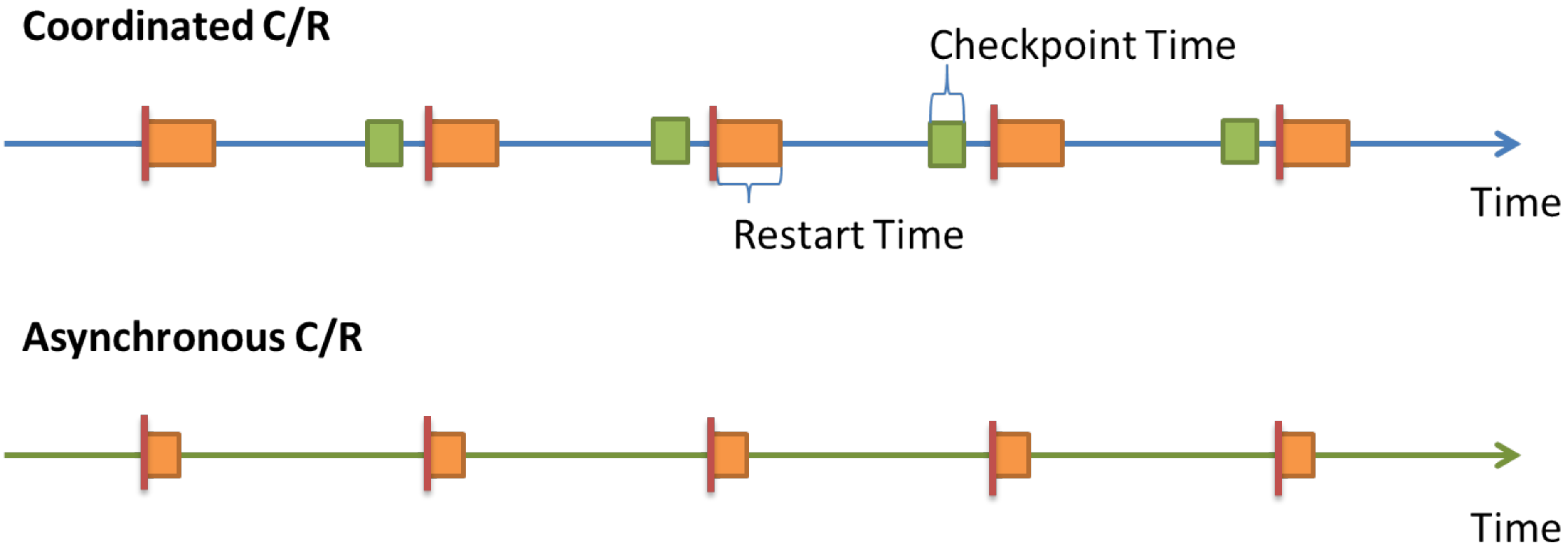
"The collection of techniques for keeping applications running to a correct solution in a timely and efficient manner despite underlying system faults"

Snir et al. (2014)

Coordinated Checkpoint/Restart



Asynchronous Checkpoint/Restart



Requirements for Asynchronous Checkpoint/Restart

Reasoning about state: Self-aware, execution frontier

Safe restart: Deterministic computation

Data race free: Monotonically increasing state

Resilience in CnC

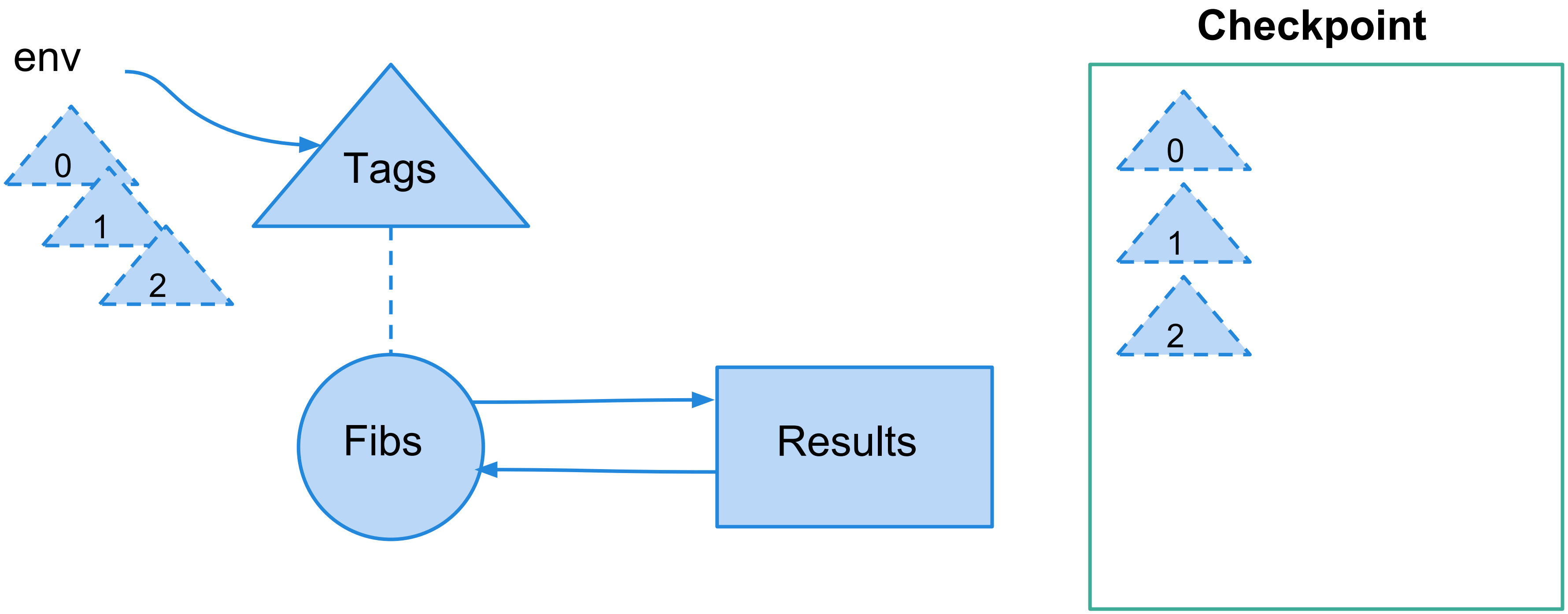
Vrvilo, N. (2014). *Asynchronous Checkpoint/Restart for the Concurrent Collections Model* (Unpublished master's thesis). Rice University, Houston, Texas USA.

Focused on shared memory CnC runtimes

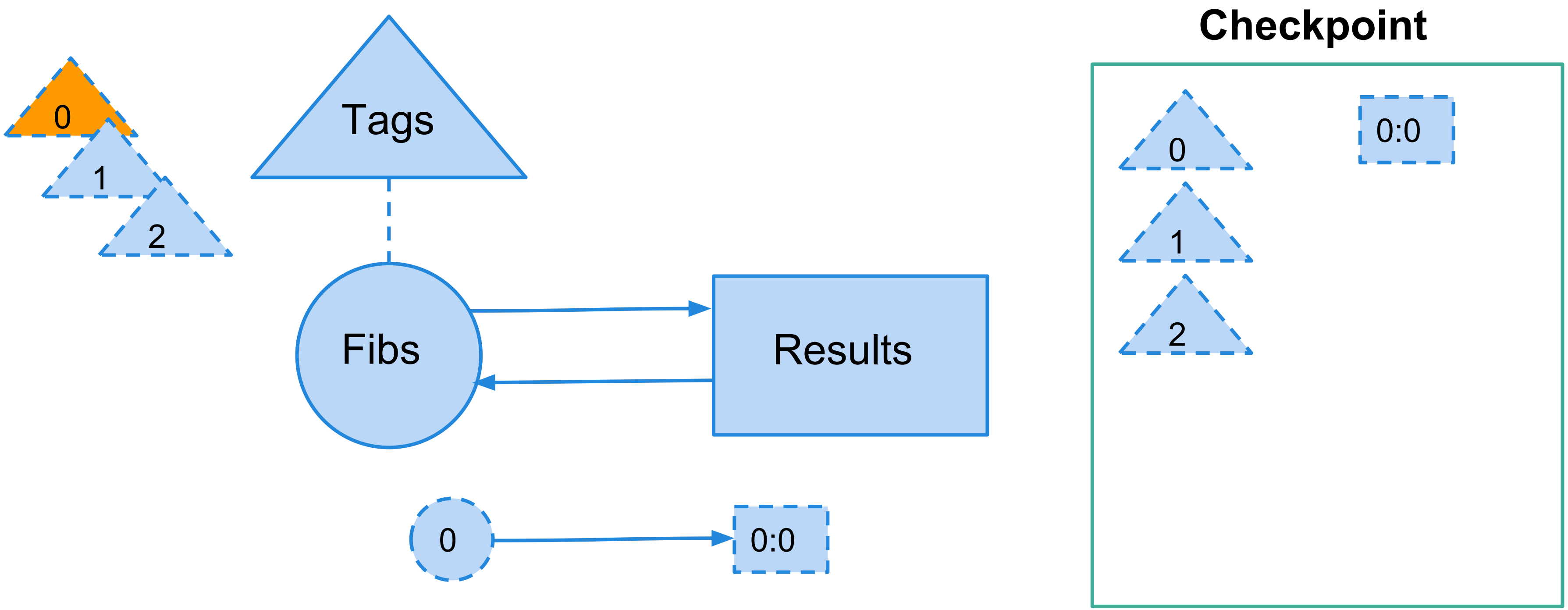
CnC Properties:

- Dependency graph
- Provable deterministic computation
- Single assignment data

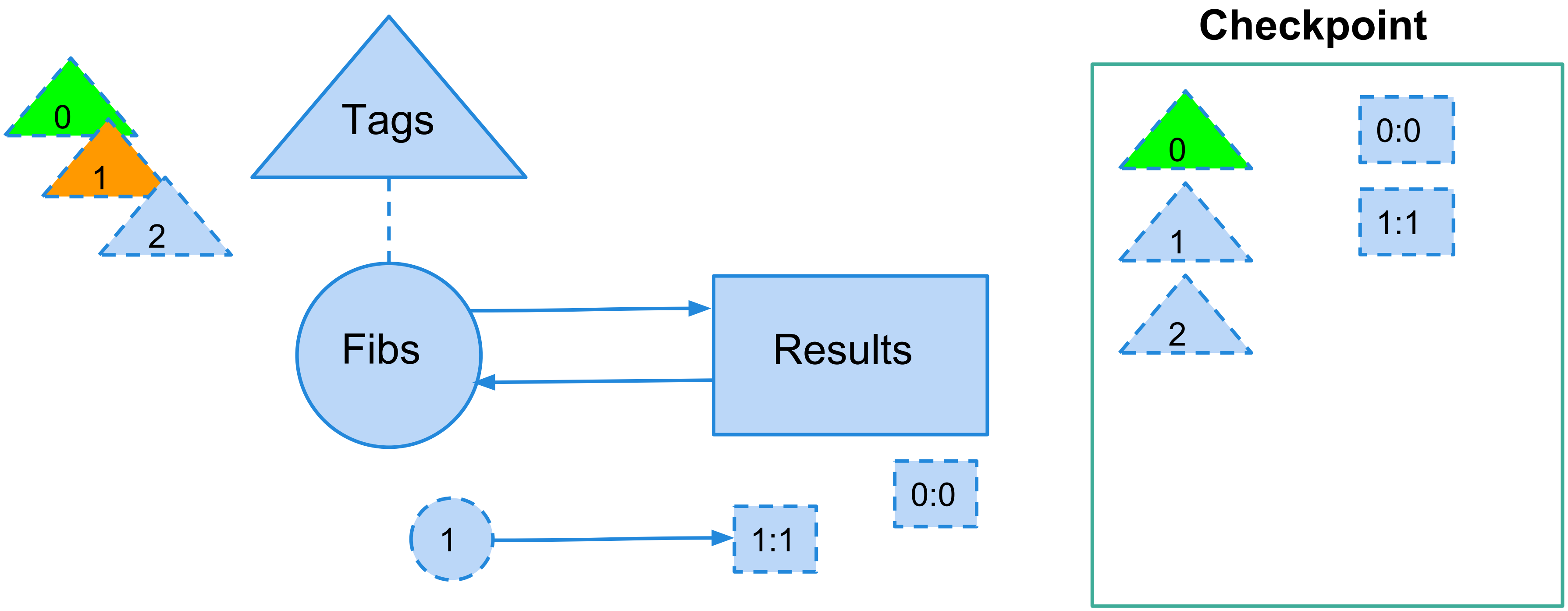
The Concurrent Collections Model



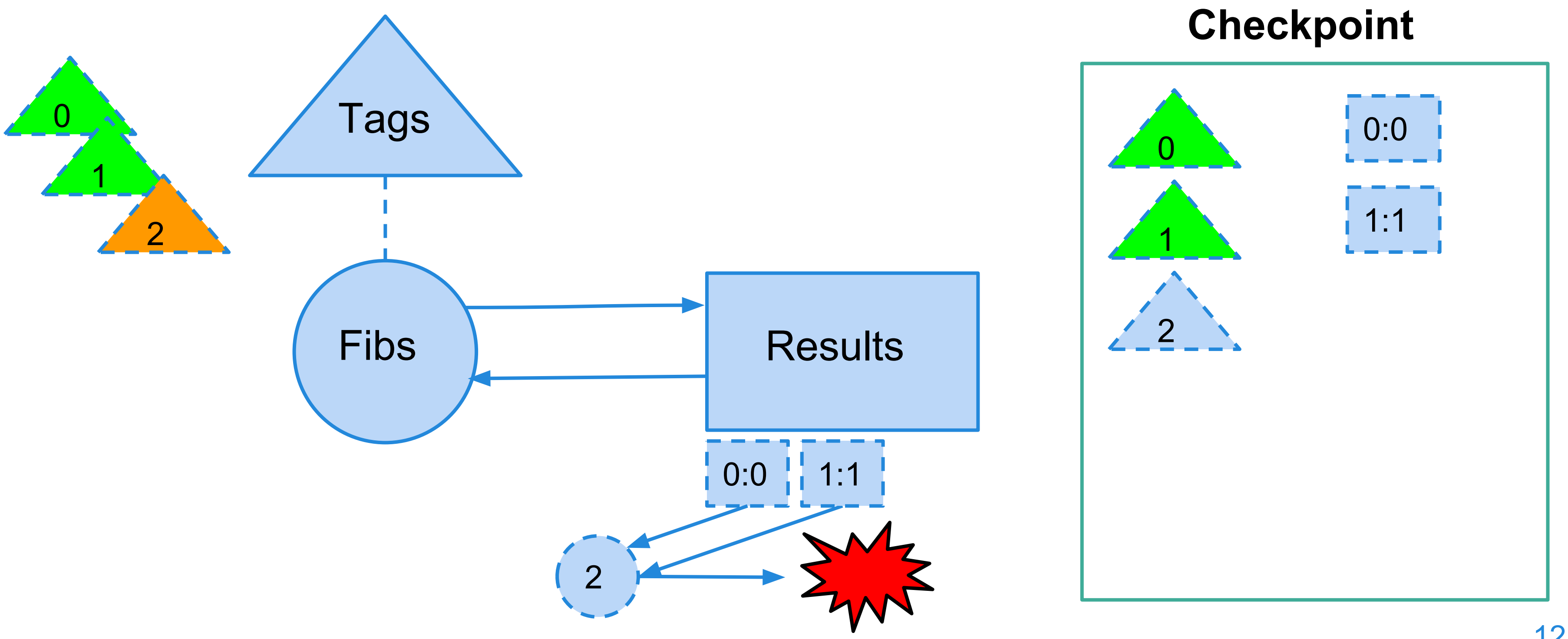
The Concurrent Collections Model



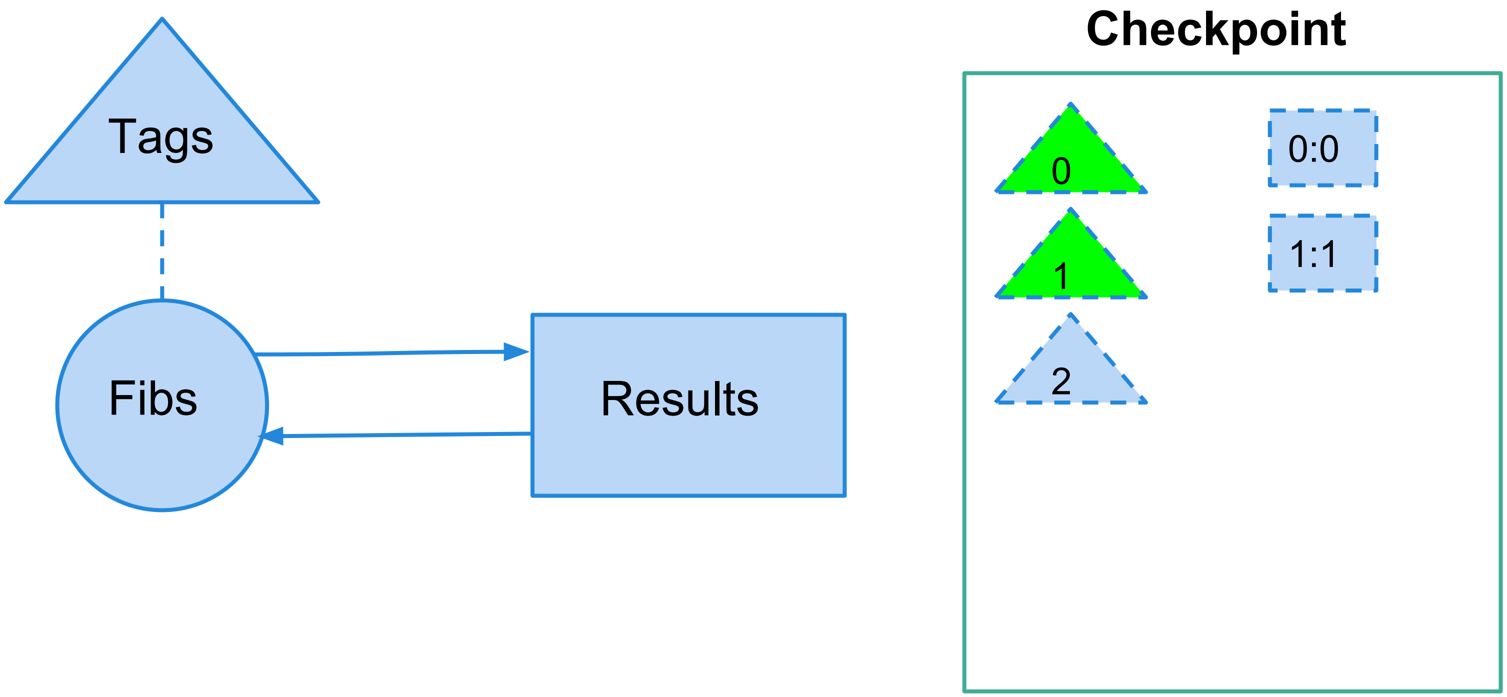
The Concurrent Collections Model



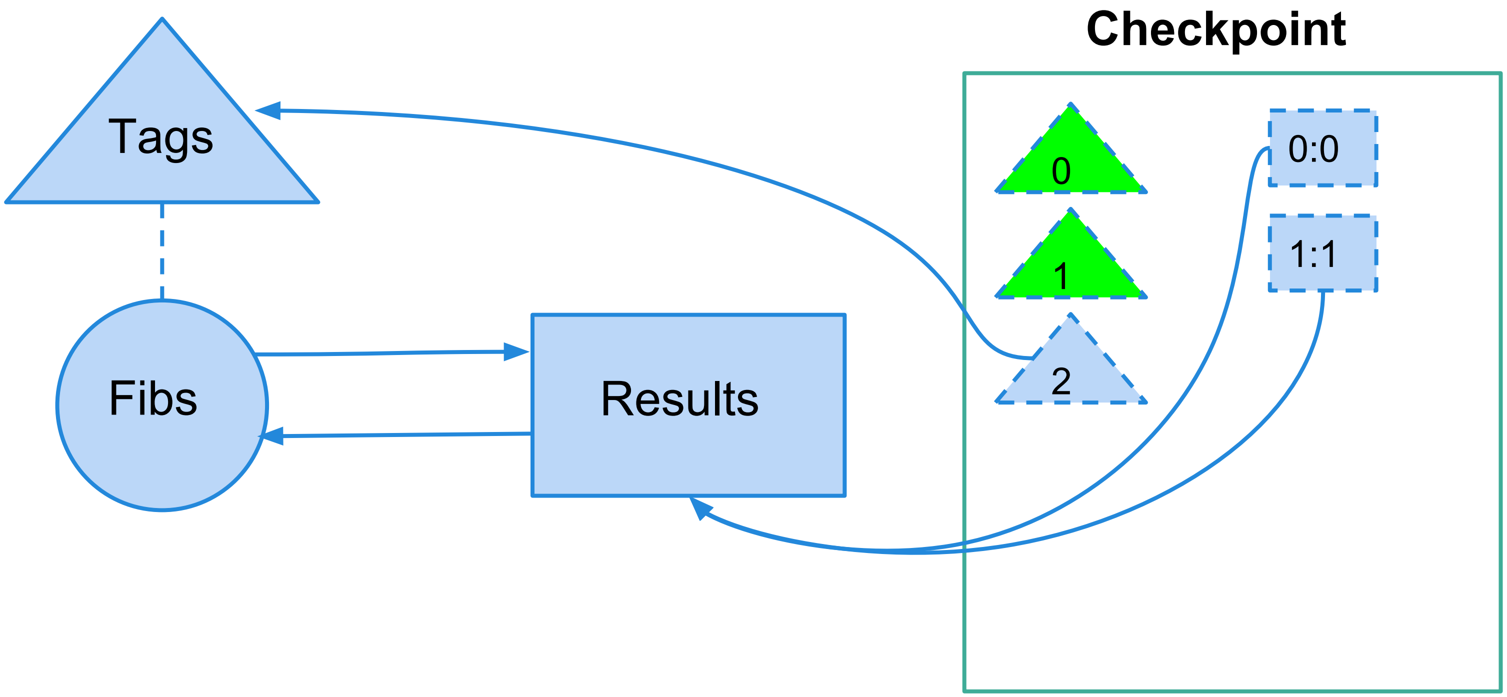
The Concurrent Collections Model



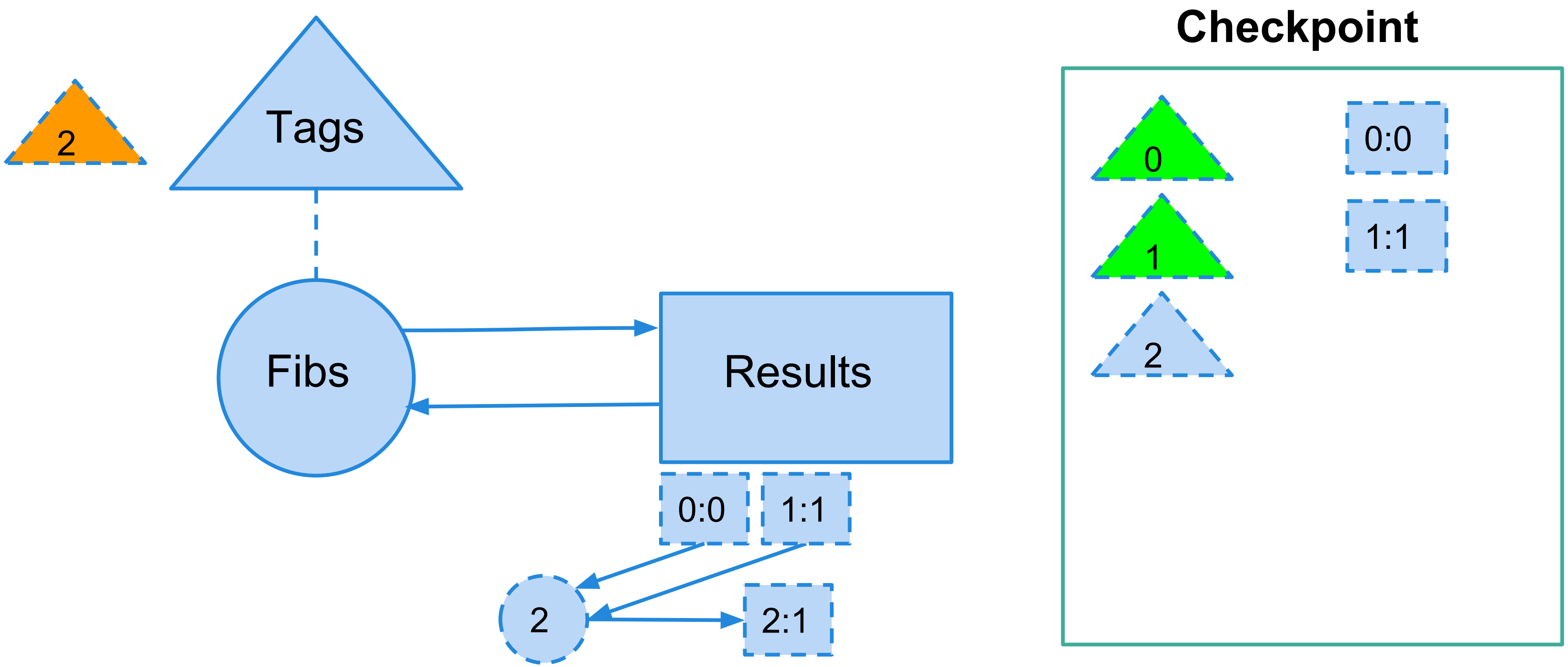
The Concurrent Collections Model



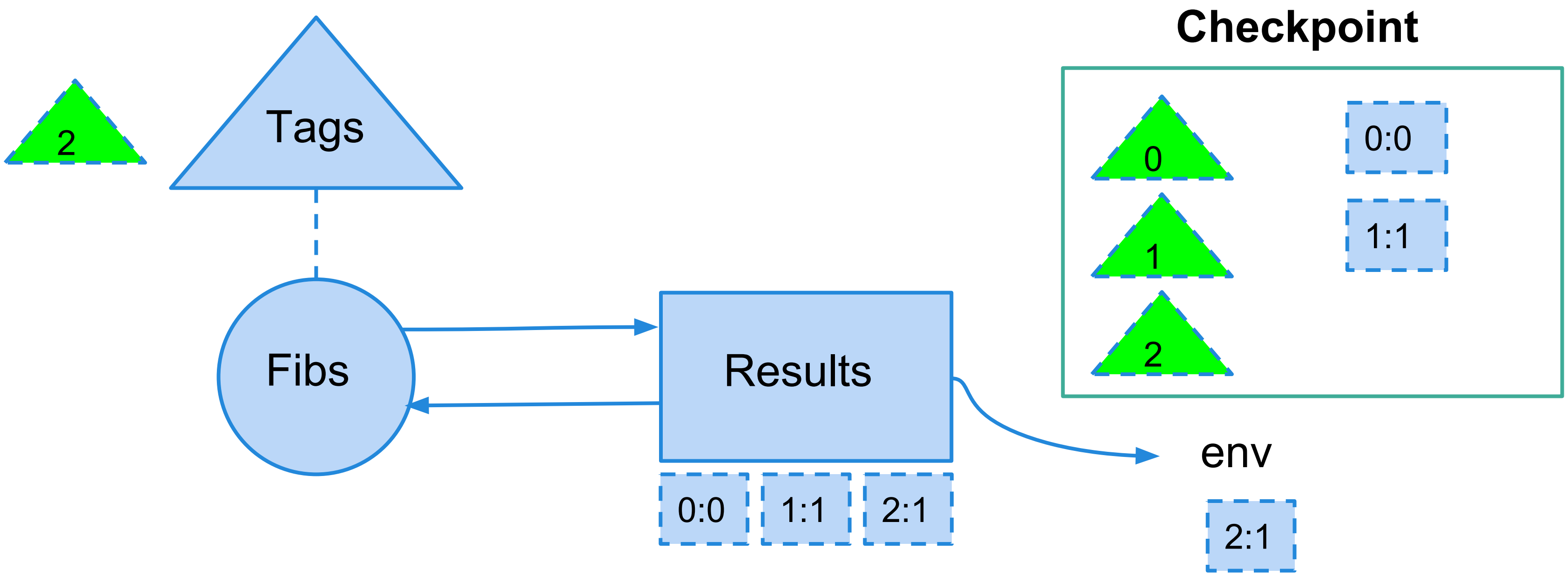
The Concurrent Collections Model



The Concurrent Collections Model



The Concurrent Collections Model



Proof of Concept Implementation

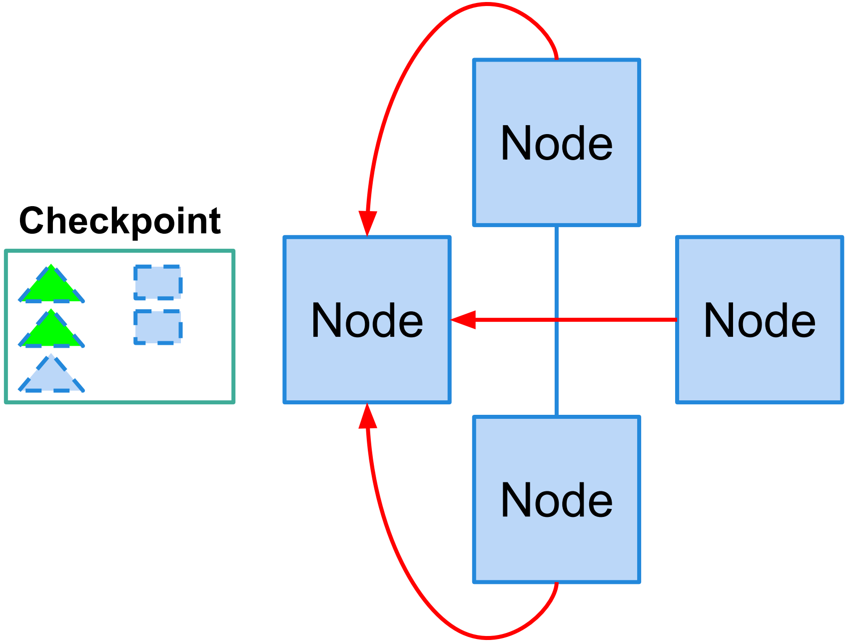
Goal: Assessing the viability of Asynchronous C/R in distributed memory CnC runtimes

Runtime: Intel(R) Concurrent Collections for C++
(Architect: Frank Schlimbach)

Resilience Flavour:

- Dedicated checkpoint node
- Fine grained updates
- Uncoordinated restart

Dedicated Checkpoint Node & Fine grained Updates



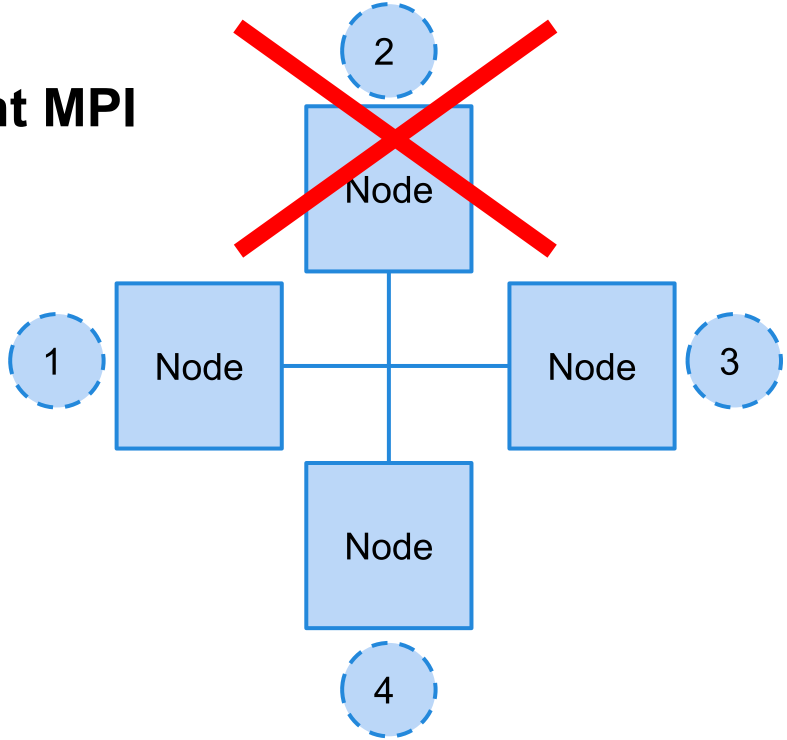
Updates contain:

data instances consumed
data instances produced
control instances produced
producers
consumers

Restart

Restart simulation ➜ No fault tolerant MPI

Uncoordinated ➜ Step duplication



Memory Management in CnC

Non-trivial: data accessed by dynamic steps

One solution: get-counting method

```
int getCountFib( FibTag t ) {
    if ( t > 0 ) {
        return 2;
    else {
        return 1;
    }
}
```

Solution

Extra bookkeeping in checkpoint:

- Consider steps only once when lowering get counts
  - Hashmap of considered steps

- Never re-add removed data instances
  - Marking data as removed

Modelling Overhead (Tw/Ts)

Coordinated Checkpoint/Restart (Daly, 2006)

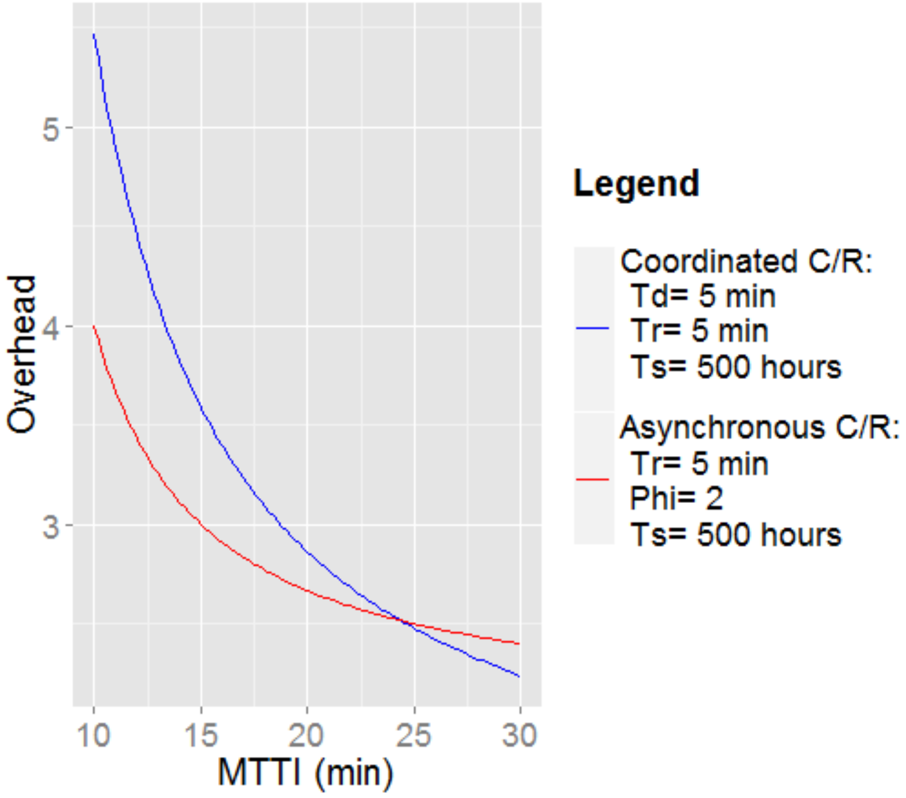
$$T_w(\tau) = M e^{T_r/M} (e^{(\tau+T_c)/M} - 1) \frac{T_s}{\tau} \quad \text{for } T_c << T_s$$

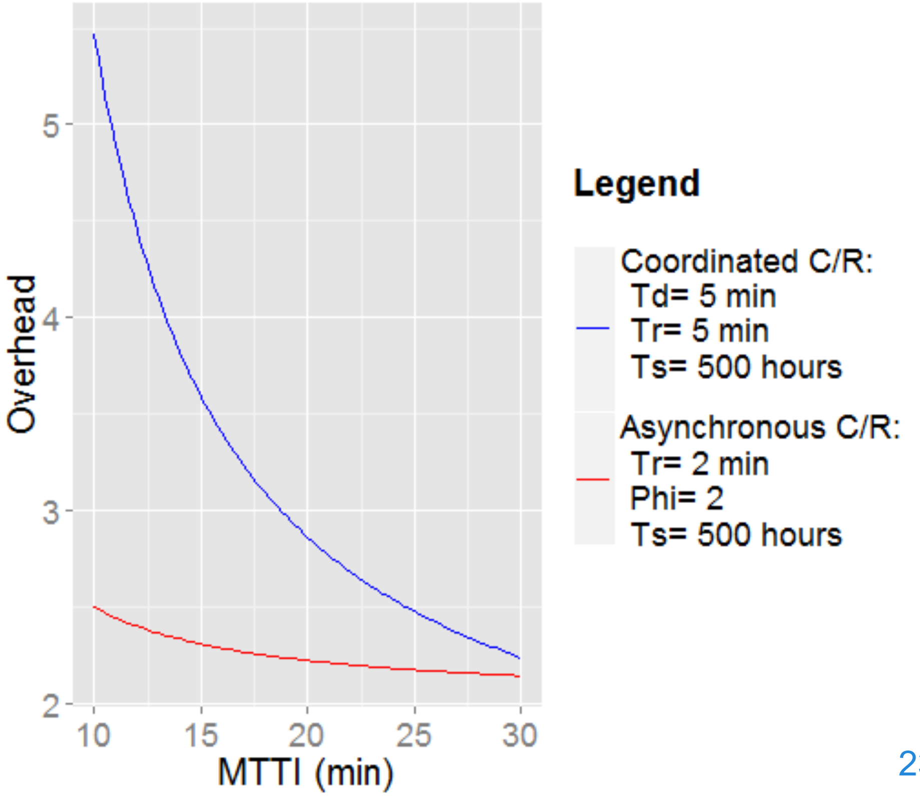
$$\tau_{opt} = \begin{cases} \sqrt{2T_c M}\left[1 + \frac{1}{3}\left(\frac{T_c}{2M}\right)^{1/2} + \frac{1}{9}\left(\frac{T_c}{2M}\right)\right] - T_c & \text{for } T_c < 2M \\ M & \text{for } T_c \geq 2M \end{cases}$$

Asynchronous Checkpoint/Restart

$$T_w(\varphi) = M\left(\frac{\varphi T_s - M}{M - T_r} + 1\right)$$

Evaluating Asynchronous Checkpoint/Restart





Benchmarks - Goals

Assessing overhead factor (φ): Ok if high

Method:

Measure w/o resilience = Solve time (Ts)

Measure with resilience = Wall clock time (Tw)

Overhead factor = Tw/Ts

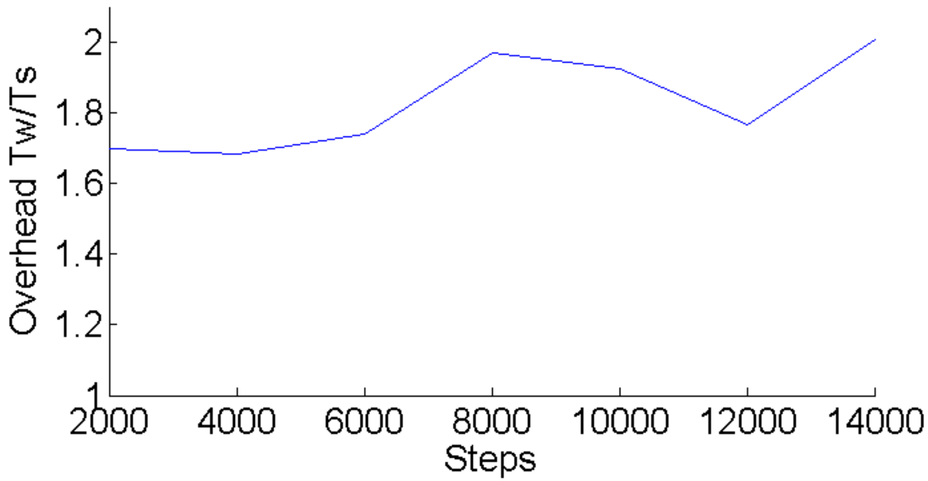
Assessing restart time (Tr): Should be low

Method:

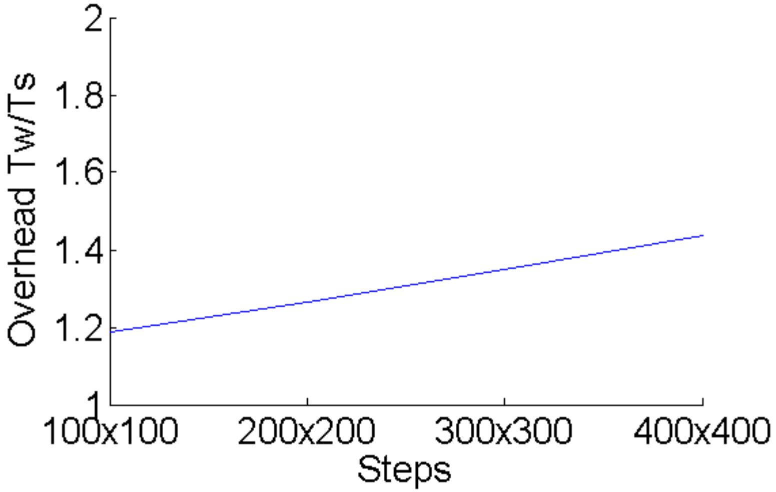
Measure time needed to calculate the restart set

Number of Steps

Fibonacci



Mandelbrot

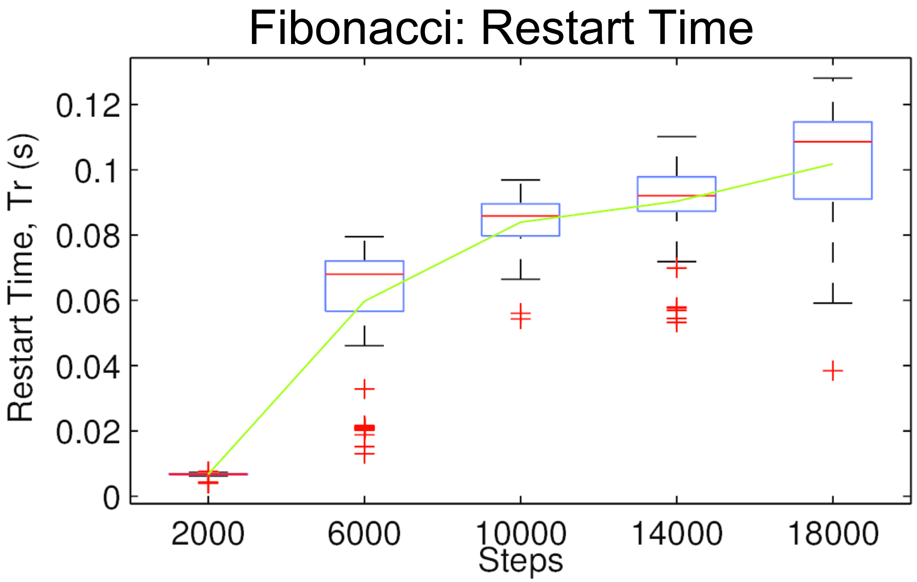


Overhead factor (φ): Increases with number of steps

Restart Time

Restart Time (Tr): Low

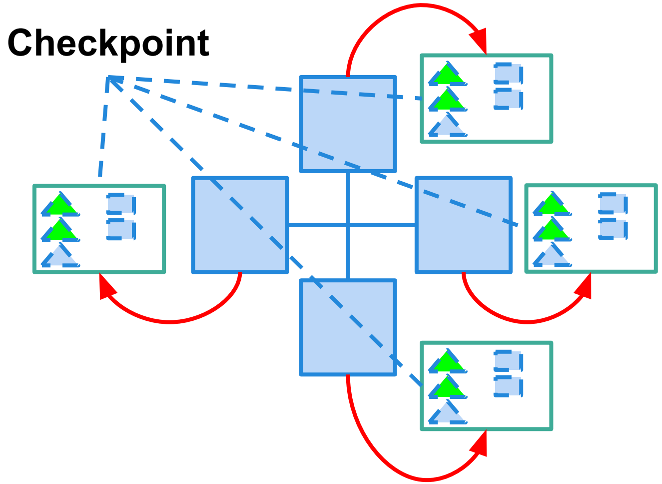
- **Optimization:** Shifting some of the complexity to the overhead factor



Future Work

Distributed Checkpoint:

- ➢ Overhead high but constant
- ➢ Restart time?



Tag-only logging:

- ➢ Less communication
- ➢ Complex restart

Conclusion

Asynchronous C/R distributed memory CnC runtime

- ➢ Analyzing different cases
- ➢ Proof of concept implementation

Asynchronous C/R is viable for systems with low SMTTI

- ➢ Model
- ➢ Proof of concept implementation

References

Daly, J. T. (2006). A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. Future Generation Computer Systems, 22(3), 303–312.

Avizienis, A., Laprie, J., Randell, B., & Landwehr, C. E. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1(1), 11–33.

Snir, M., Wisniewski, R. W., Abraham, J. A., Adve, S. V., Bagchi, S., Balaji, P., . . . Hensbergen, E. V. (2014). Addressing Failures in Exascale Computing. International Journal of High Performance Computing Applications, 28(2), 129–173.

Franck Cappello (2009). Fault Tolerance in Petascale/ Exascale Systems: Current Knowledge. *International Journal of High Performance Computing, 23(1)*, 212-226.

Vrvilo, N. (2014). *Asynchronous Checkpoint/Restart for the Concurrent Collections Model* (Unpublished master's thesis). Rice University, Houston, Texas USA.