Toward Exascale Resilience

Part 8: Containment Domains / cross-layer schemes

Mattan Erez
The University of Texas at Austin

July 2015

Credit to:

UT Austin students:

- Benjaming Cho, Jinsuk Chung, Ali Fakhrzadehgan, Ikhwan Lee, Kyushick Lee, Seong-Lyong Gong, Mike Sullivan, Song Zhang, Doe Hyun Yoon (now at Google)

Collaborators (growing list)

- Cray, NVIDIA, ETI
- LBNL: Brian Austin, Dan Bonachea, Paul Hargrove , Sherry Li, Eric Roman

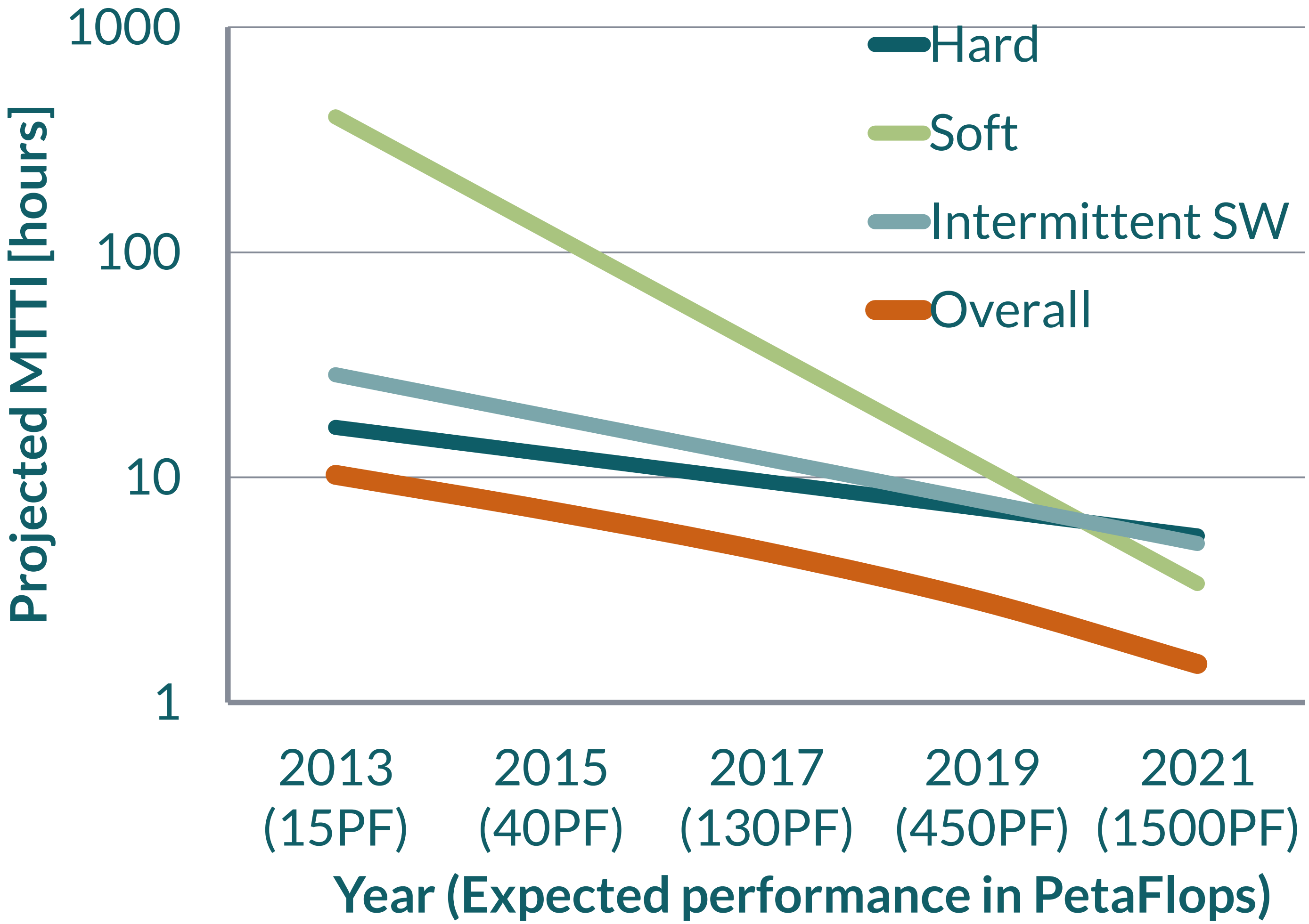
Funding agencies

- DOE ECRP, XStack, FF, PSAAP II
- Initial funding from DARPA UHPC

The **constraints**:

- Power/energy
- Time
- Money
- Correctness

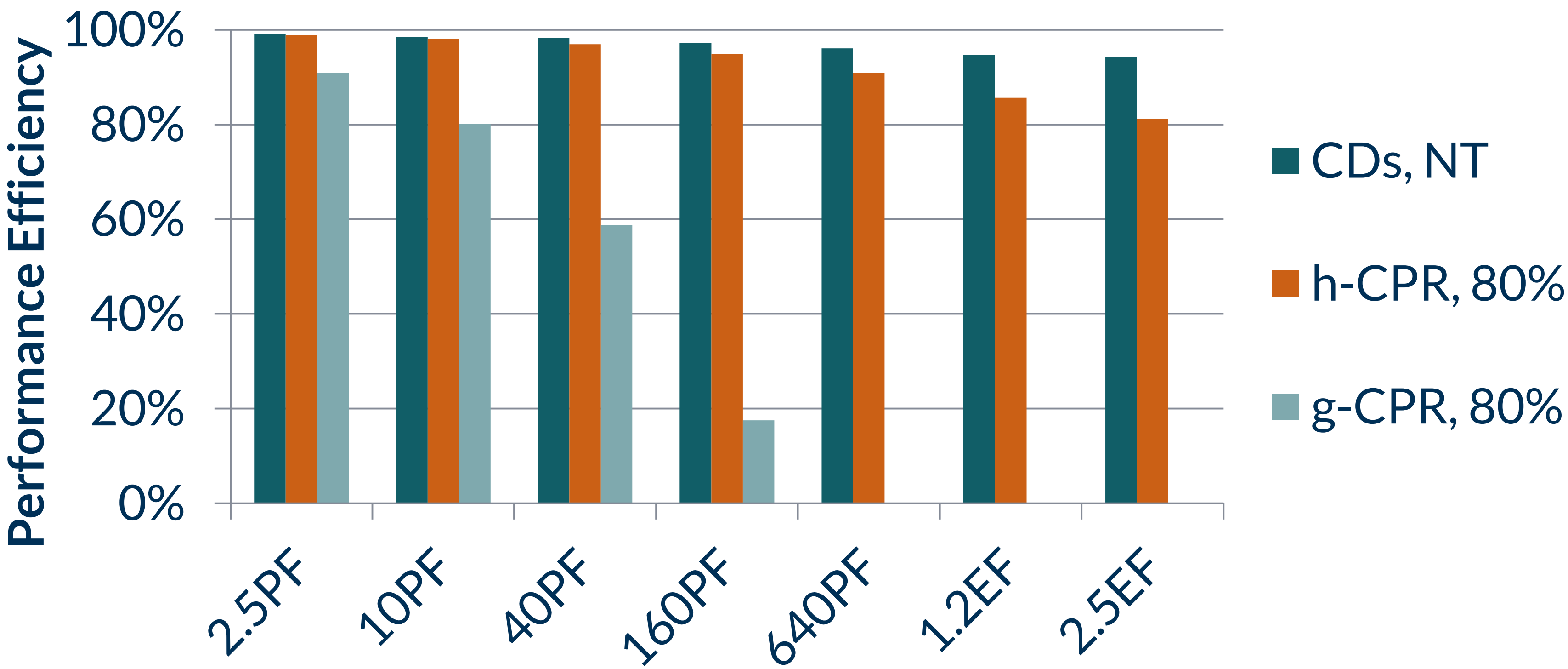
Resilience is a big challenge for **DOE computations**



Something **bad** every ~**minute at DOE** scale

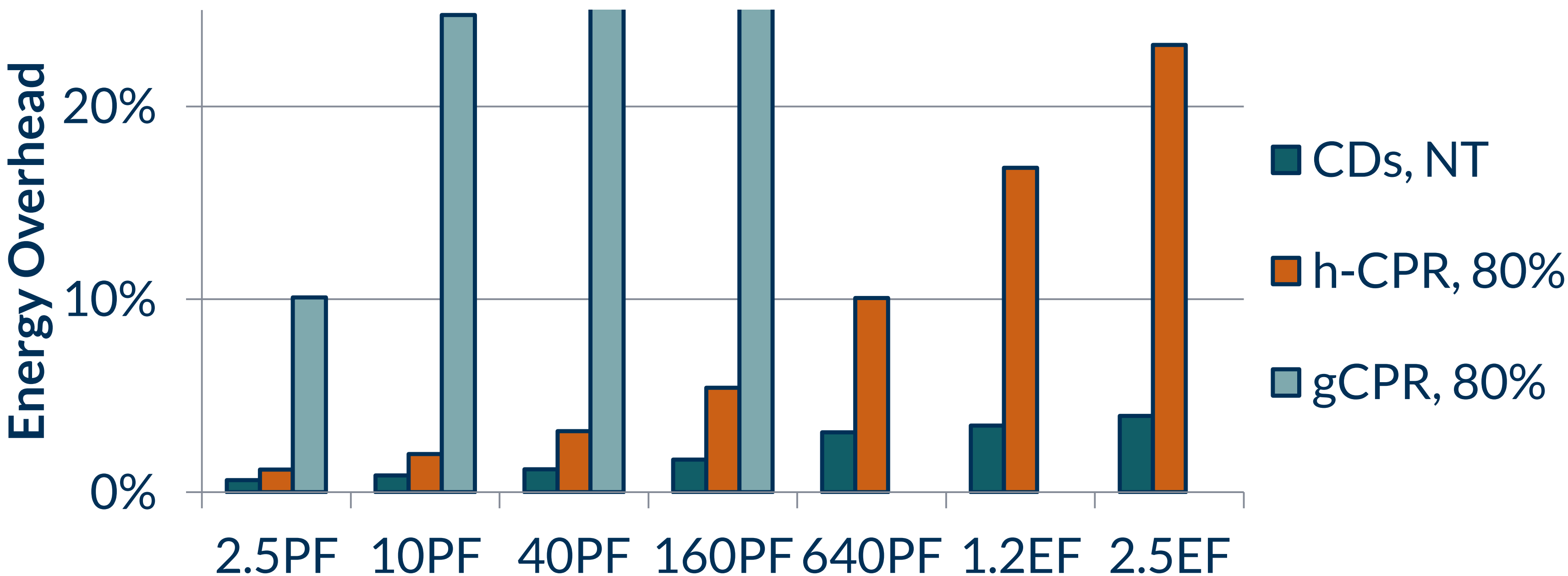
The baseline: **checkpoint-restart**

Not good enough on its own



Failure rate too high for checkpoint/restart

Correctness also at risk



Energy also problematic

The **cost** of resilience

- Preparation
- Detection
- Mitigation (repair + recover)
- Implementation

Software?

Hardware?

Algorithm?

Software?

Hardware?

Algorithm?

Containment Domains:

adaptive holistic approach

– Per-experiment balance of energy, time, money, correctness

Can **hardware alone** solve the problem?

Yes, but **costly**

- **Significant** and likely **fixed** overheads
- May not be needed in many commercial settings

Fixed overhead examples (estimated)

Both energy and/or throughput

- Up to ~25% chipkill **correct vs.** chipkill **detect**
- 20 – 40% for pipeline SDC reduction
- >2X for arbitrary correction
- Even greater overhead if protecting approximate units

Something bad every ~**minute at DOE**

Something bad every **year commercially**

- Smaller units of execution
- Different requirements

Locality and hierarchy are key

- Hierarchical constructs
- Distributed operation

Range of correctness requirements

What about **algorithmic resilience**?

- Algorithmic detection
- Iterative converging algorithms
- Redundant information
- Probabilistic methods

Examples on board

- Algorithmic check of matrix multiplication
- Algorithmic check of a solver
- Convergent calculation
  - Simple and basic Newton-Raphson
- Monte Carlo

But,

Different apps → different techniques

Different scales → different techniques

Need to adapt/co-tune

Containment Domains
elevate resilience to **first-class abstraction**

- Program-structure abstractions
- Composable resilient program components
- Regimented development flow
- Supporting tools and mechanisms

Containment Domains

- **Abstract** resilience constructs that span system layers
- **Hierarchical and Distributed** operation for locality
- **Scalable** to large systems with high energy efficiency
- **Heterogeneous** to match disparate error/failure effects
- **Proportional** and effectively balanced
- **Tunable** resilience specialized to application/system
- **Analyzable** and auto-tuned

CDs Embed Resilience within Application

Express resilience as a tree of CDs

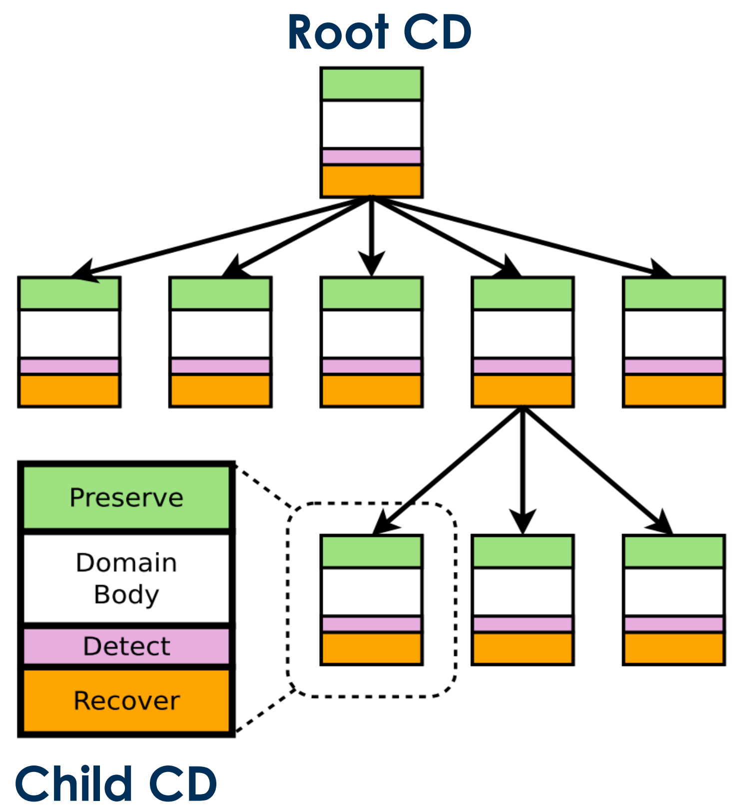
- Match CD, task, and machine hierarchies
- Escalation for differentiated error handling

Semantics

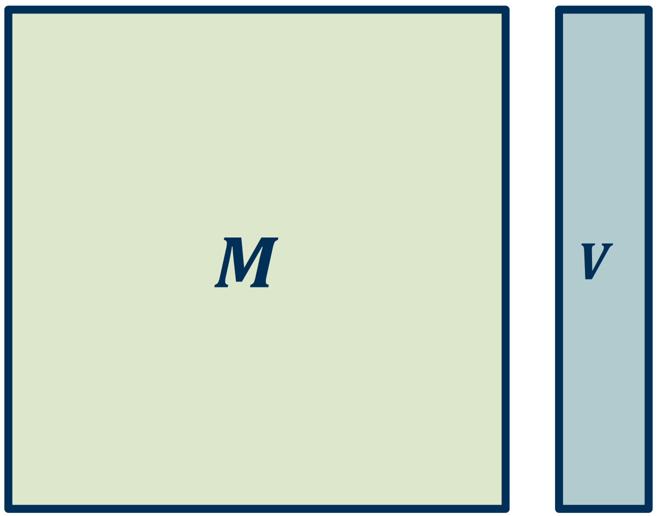
- Erroneous data never communicated
- Each CD provides recovery mechanism

Components of a CD

- ***Preserve*** data on domain start
- ***Compute*** *(domain body)*
- ***Detect*** faults before domain commits
- ***Recover*** from detected errors



Mapping example: SpMV



Matrix **M** Vector **V**

```
void task<inner> SpMV(in M, in Vi, out
                      Ri) {
   cd = GetCurrentCD()
           ->CreateAndBegin();
   cd->Preserve(matrix, size, kCopy);
   forall(…) reduce(…)
       SpMV(M[…],Vi[…],Ri[…]);
   cd->Complete();
}
```

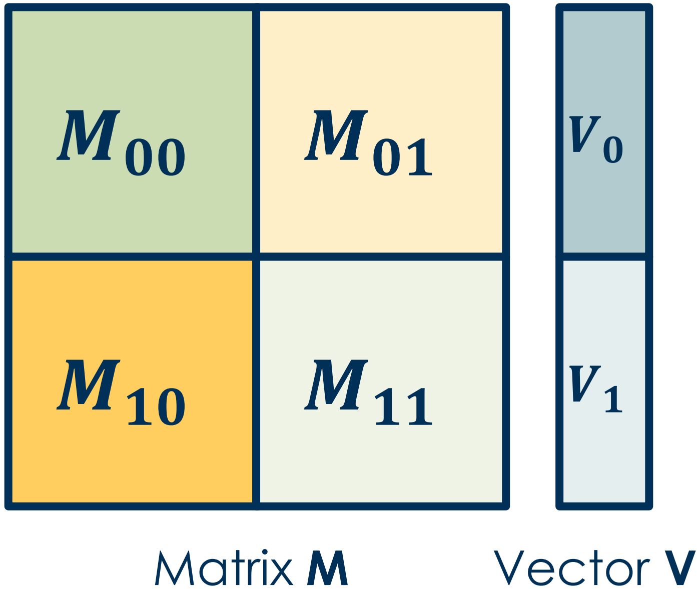
```
void task<leaf> SpMV(…) {
 cd = GetCurrentCD()
           ->CreateAndBegin();
 cd->Preserve(M, sizeof(M), kRef);
 cd->Preserve(Vi, sizeof(Vi), kCopy);
 for r=0..N
  for c=rowS[r]..rowS[r+1]
    resi[r]+=data[c]*Vi[cIdx[c]];
    cd->CDAssert(idx > prevIdx,
kSoft);
    prevC=c;
 cd->Complete();
}
```

Mapping example: SpMV

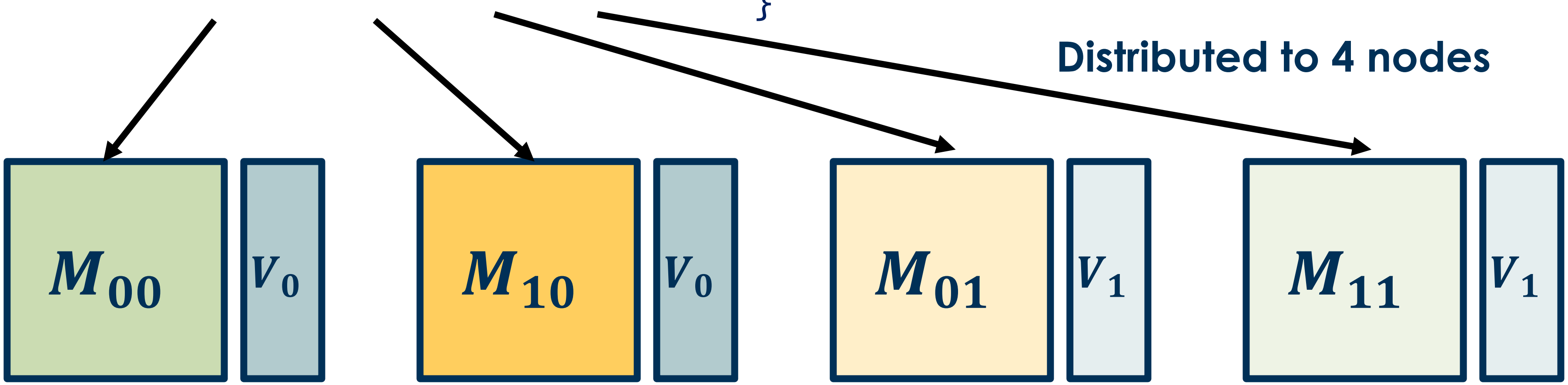

```
void task<inner> SpMV(in M, in Vi, out
                      Ri) {
   cd = GetCurrentCD()
           ->CreateAndBegin();
   cd->Preserve(matrix, size, kCopy);
   forall(…) reduce(…)
       SpMV(M[…],Vi[…],Ri[…]);
   cd->Complete();
}
```

```
void task<leaf> SpMV(…) {
 cd = GetCurrentCD()
           ->CreateAndBegin();
 cd->Preserve(M, sizeof(M), kRef);
 cd->Preserve(Vi, sizeof(Vi), kCopy);
 for r=0..N
  for c=rowS[r]..rowS[r+1]
    resi[r]+=data[c]*Vi[cIdx[c]];
    cd->CDAssert(idx > prevIdx,
kSoft);
    prevC=c;
 cd->Complete();
}
```

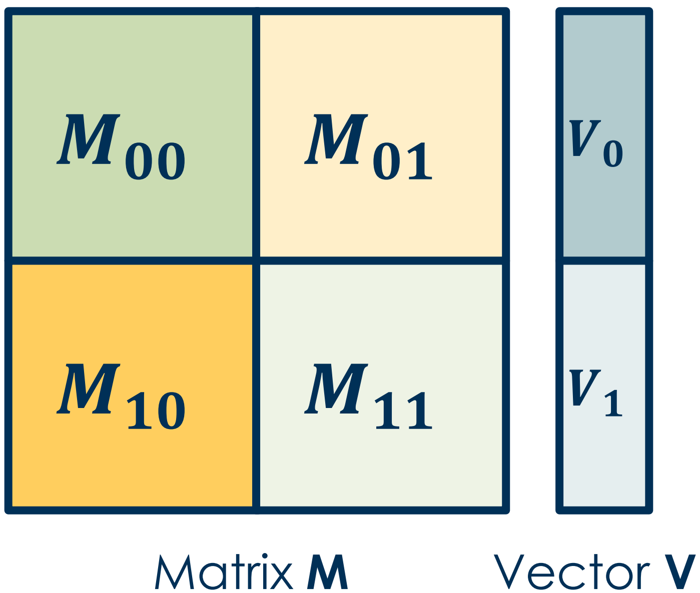
Mapping example: SpMV



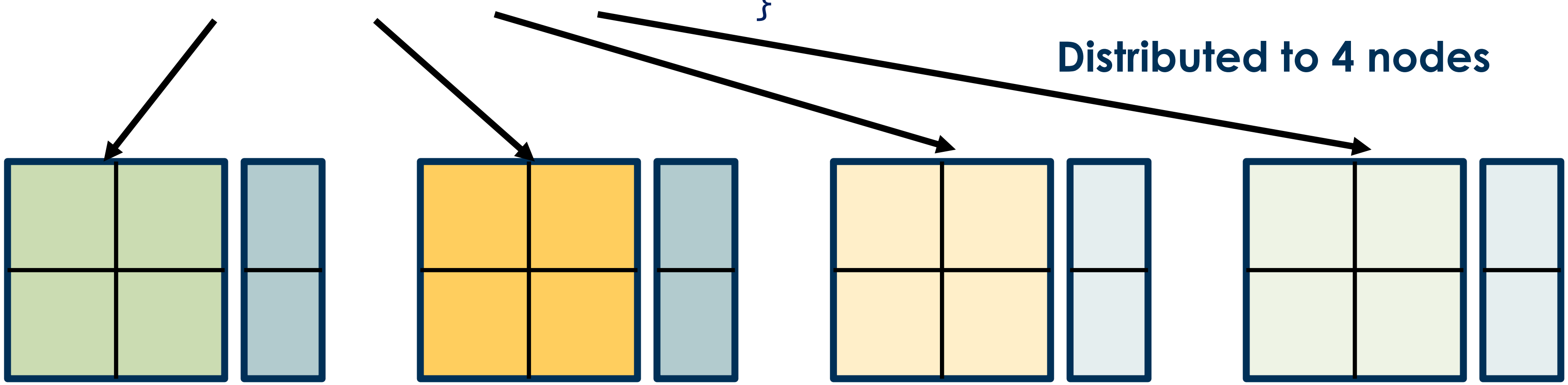

```
void task<leaf> SpMV(…) {
 cd = GetCurrentCD()
          ->CreateAndBegin();
 cd->Preserve(M, sizeof(M), kRef);
 cd->Preserve(Vi, sizeof(Vi), kCopy);
 for r=0..N
  for c=rowS[r]..rowS[r+1]
    resi[r]+=data[c]*Vi[cIdx[c]];
    cd->CDAssert(idx > prevIdx,
kSoft);
    prevC=c;
 cd->Complete();
}
```



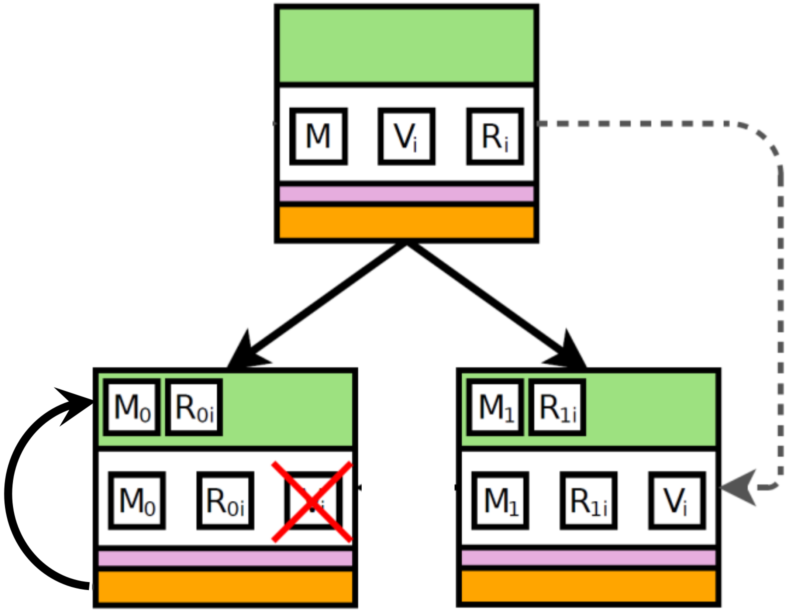
Mapping example: SpMV




```
void task<leaf> SpMV(…) {
 cd = GetCurrentCD()
           ->CreateAndBegin();
 cd->Preserve(M, sizeof(M), kRef);
 cd->Preserve(Vi, sizeof(Vi), kCopy);
 for r=0..N
  for c=rowS[r]..rowS[r+1]
    resi[r]+=data[c]*Vi[cIdx[c]];
    cd->CDAssert(idx > prevIdx,
kSoft);
    prevC=c;
 cd->Complete();
}
```

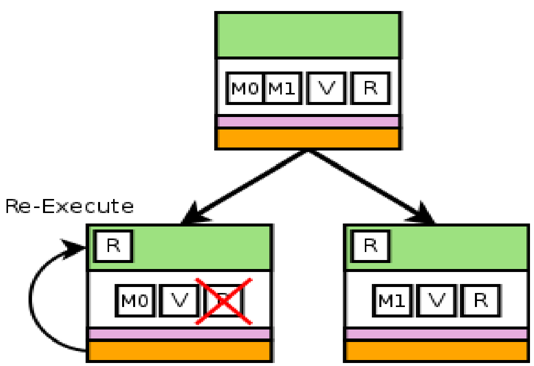


Concise abstraction for complex behavior

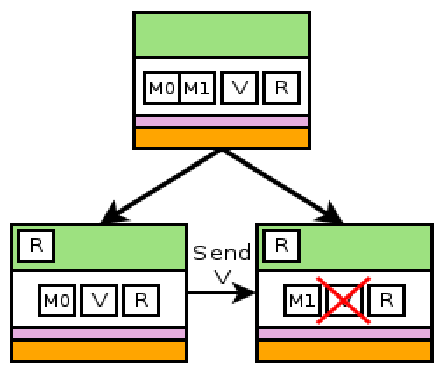



```
void task<leaf> SpMV(…) {
 cd = GetCurrentCD()
          ->CreateAndBegin();
 cd->Preserve(M, sizeof(M), kRef);
 cd->Preserve(Vi, sizeof(Vi), kCopy);
 for r=0..N
  for c=rows[r]..rows[r+1]
    resi[r]+=data[c]*Vi[cIdx[c]];
    cd->CDAssert(idx > prevIdx,
kSoft);
    prevC=c;
 cd->Complete();
}
```

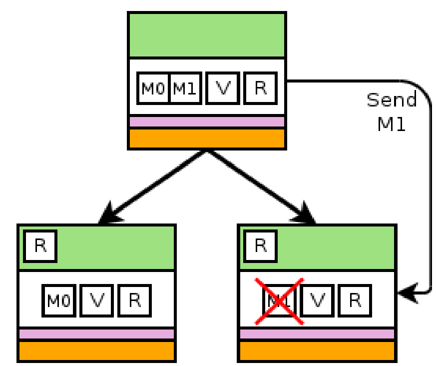
Local copy or regen



Sibling



Parent (unchanged)



Programming and execution model support

CDs manage preservation, restoration, and re-execution

- Allocate and frees storage
- Transfer data
- Manage default error detection
- Call appropriate CD (hierarchy level) on error/fault
- Holistic error reporting

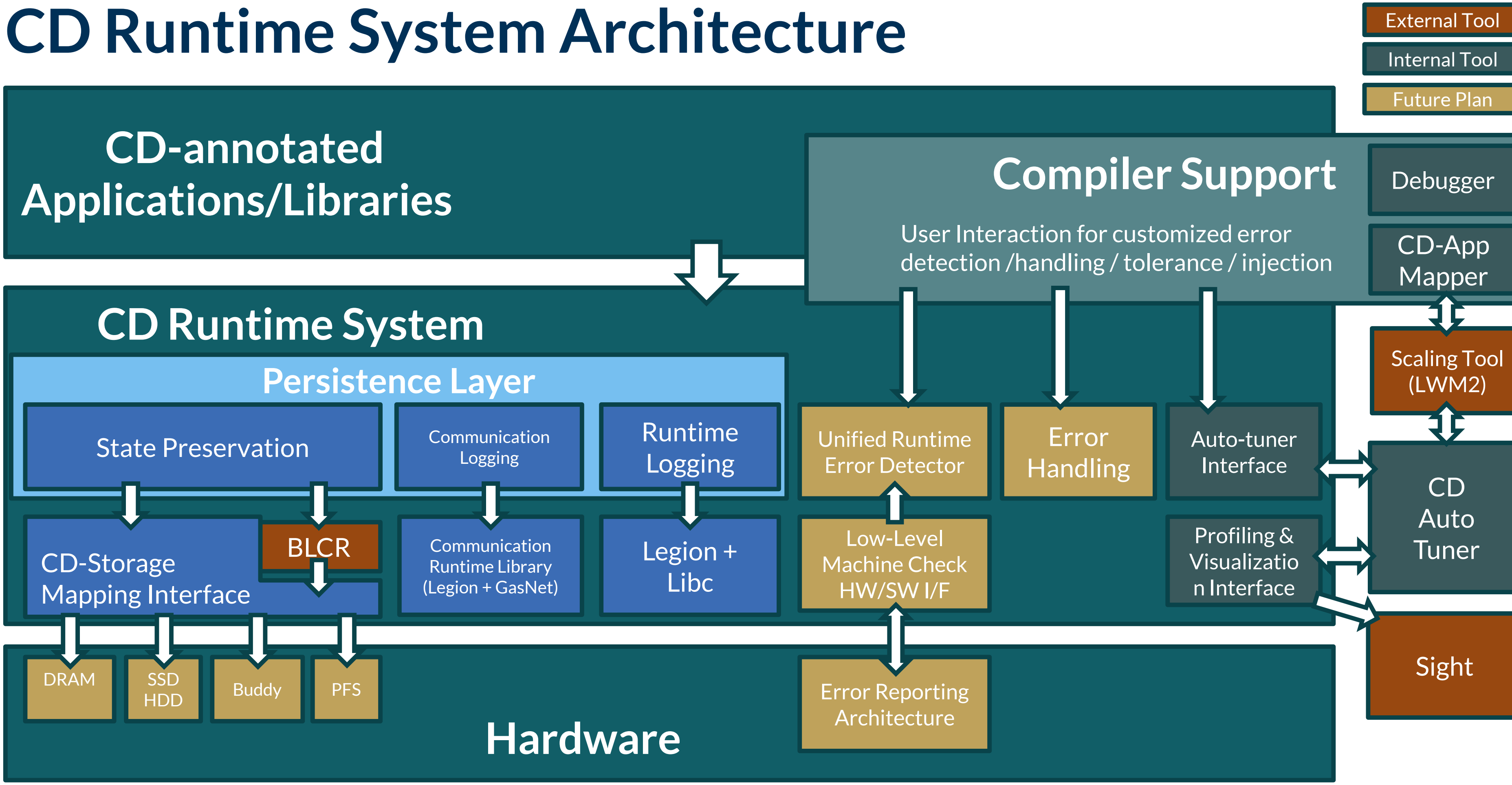
Specific policies can be written by the user

- Specialize and tune every aspect of resilience
- Straightforward abstractions

CD abstraction amenable to analysis and auto-tuning

- Analytical model fed with application properties

CD Runtime System Architecture



– Annotations, persistence, reporting, recovery, tools

CD usage flow

- Annotate
- Profile and extrapolate CD tree
- Supply machine characteristics
- Analyze and auto-tune
  - Flexible preservation, detection, and recovery
- Refine tradeoffs and repeat
- Execute and monitor
  - CD management and coordination
  - Distributed and hierarchical preservation
  - Distributed and hierarchical recovery

CD annotations express **intent**

- **CD hierarchy** for scoping and consistency
- **Preservation** directives and hints exploit locality
- **Correctness** abstractions
  - Detectors and tolerances
- **Recovery** customization
- **Debug/test** interface

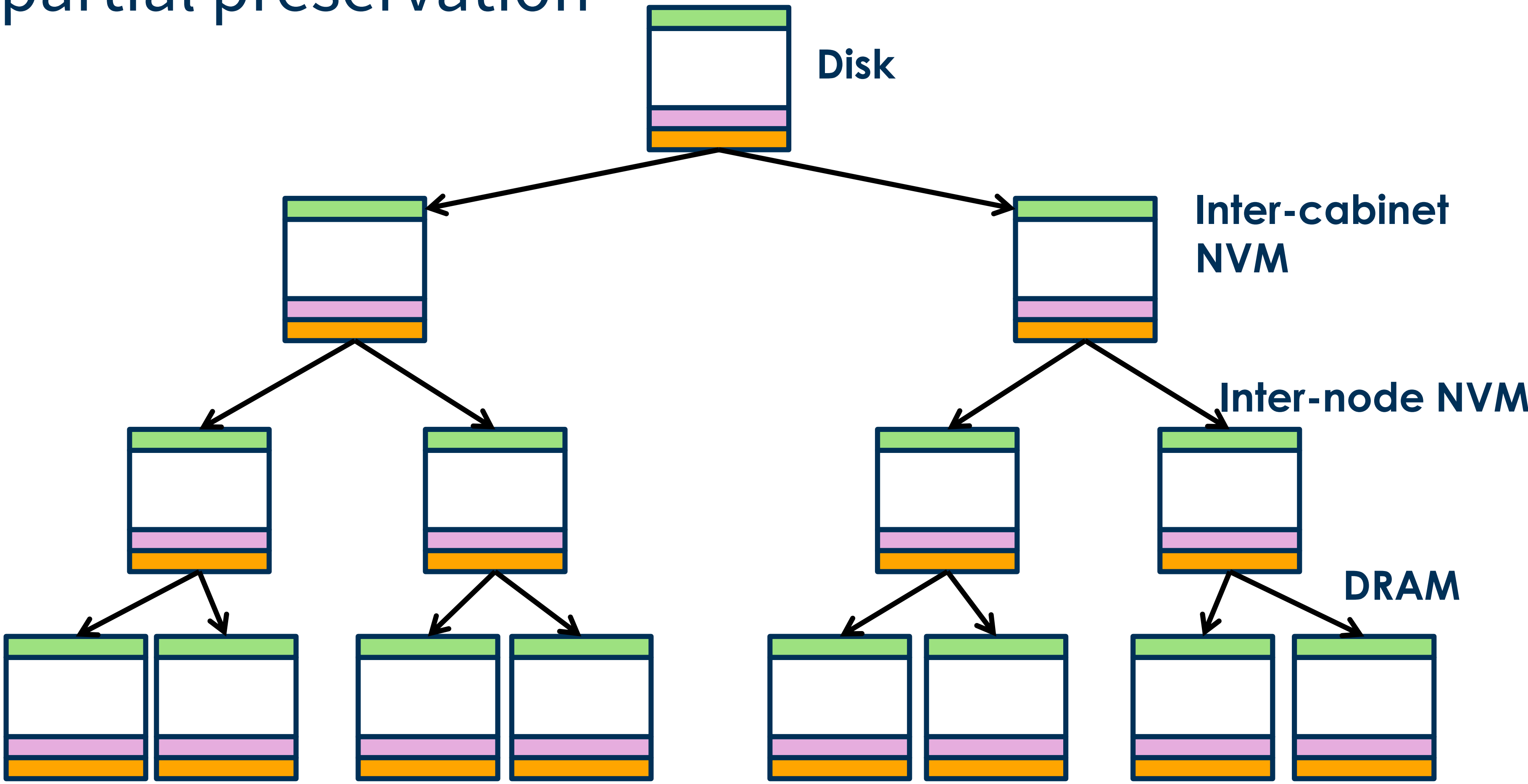
Work in progress: http://lph.ece.utexas.edu/users/CDAPI

State preservation and restoration API

```
curCD->Preserve(ptr,size,method_mask,
                byref_name, name, regenObj);
```

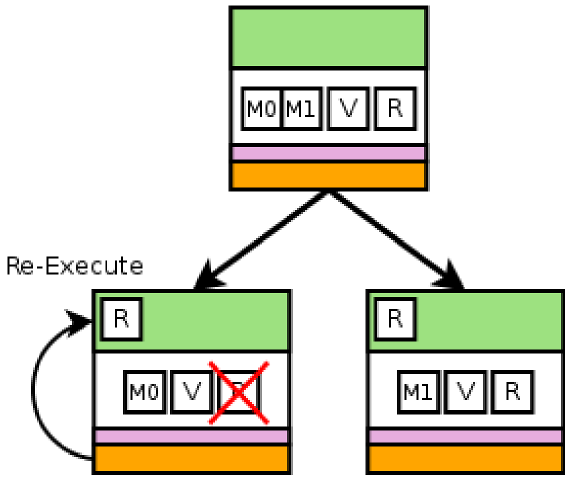
- Hierarchical
  - Per CD (level)
  - Match storage hierarchy
  - Maximize locality and minimize overhead
- Proportional
  - Preserve only when worth it (skip preserve calls)
  - Exploit inherent redundancy
  - Utilize regeneration

Hierarchical local recovery and partial preservation

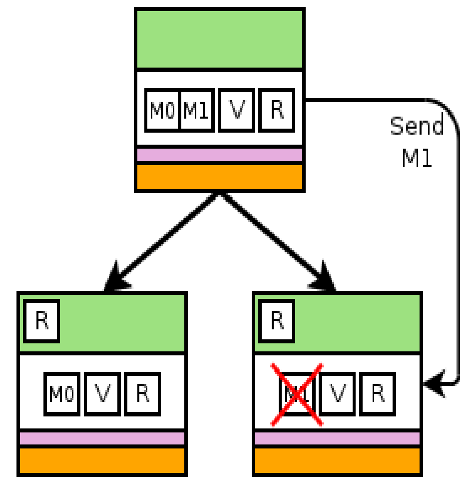


Partial preservation via sibling, parent, or regeneration where appropriate

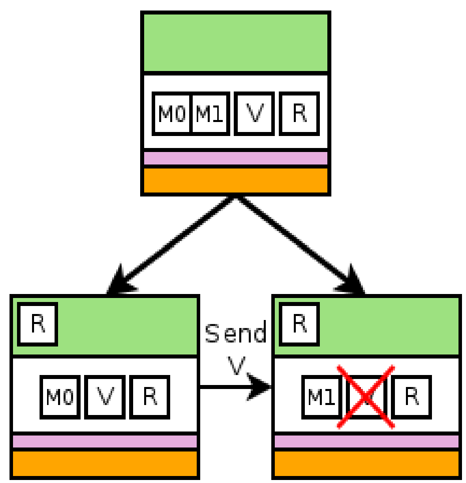
Local copy or regen



Parent (unchanged)



Sibling



Correctness abstractions

- Detectors
- Requirements
- Recovery

What can go **wrong**?

- Application crash
- Process crash
- Process unresponsive
- Failed communication
- Hardware
  - Cache error
  - Memory error
  - TLB error
  - Node offline
  - …

What can go **wrong**?

- Lost resource
- Wrong value
  - Specific address?
  - Specific access?
  - Specific computation?
- Degraded resource

Who detects?

How reported?

Today: machine check architecture

- (Maskable) interrupts
- Complex encoding of errors / failures
  - Spread across many processor-specific state registers
  - Very difficult to parse and use
- Currently – **level of containment** reported
  - **Enables fine-grained software recovery**
  - Know before state is corrupted
  - Know when only process state is corrupted
- Event counters and triggers for errors
  - Root cause analysis

Today: machine check architecture

- Not suitable for programmers
  - Barely suitable for system implementers
  - Doable, but tricky and requires a lot of reading
- Varies by vendor
- Continuously updated

System-provided detectors

– `curCD->Detect();`
  - Control response granularity

User-specified detectors

– `curCD->`
  `CDAssert(test, error_to_report);`

Consistent and unified reporting & analysis

Catch the error as soon as possible

- Less to recover
- Ideally smaller and faster preservation
- Micro-rollbacks
- Idempotent regions
- Hardware-level rollbacks

- Idempotent regions and hardware-rollback
  - What if hardware can automatically rollback and rexecute?
    - Fine-grained recovery will have little impact on performance
    - Users may not need to do anything

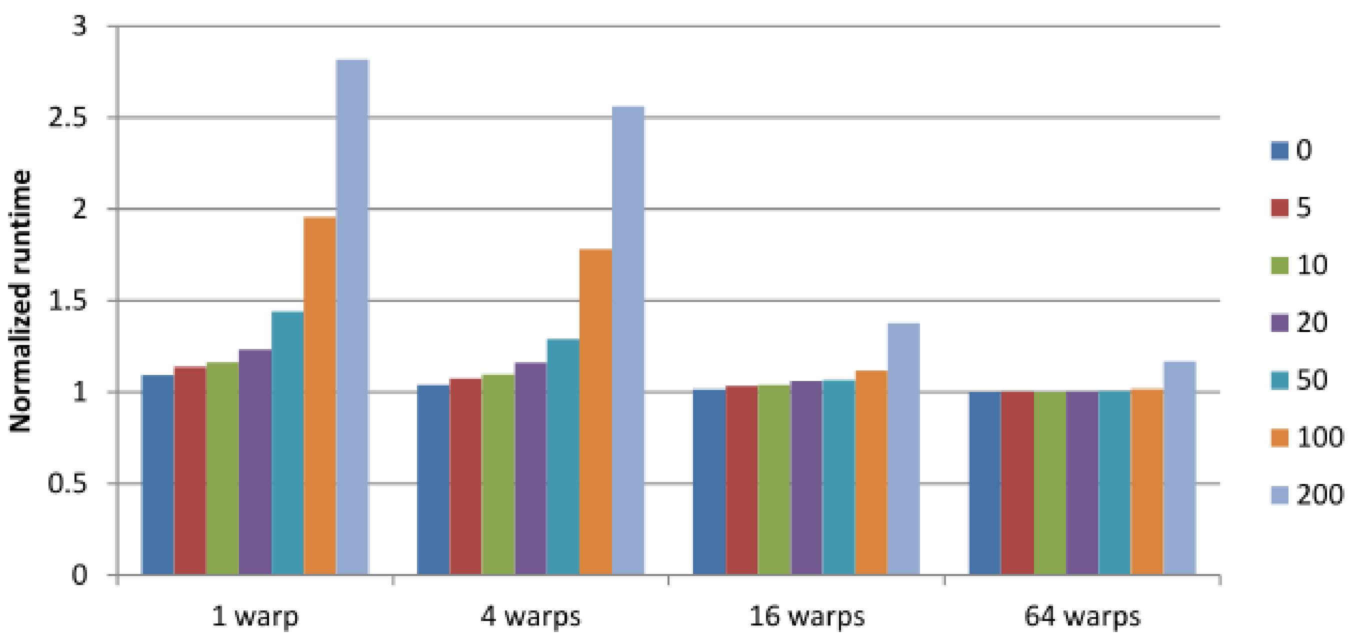
- Instruction retry
  - Out-of-order processors
  - In-order and GPUs?

Sophisticated out-of-order offer ample opportunity for hardware retry

- Speculative execution can be used to recovery from soft errors
- ROB and LSQ buffer temporary results
- Transactional memory does to

Harder in a GPU

- Need to ensure effect-free rollback
  - No hardware buffering
- Idempotent regions and CDs
- Tradeoffs with hardware buffering and detection latency



Express **correctness intent**

- ```
  curCD->
    RegisterDetection(errors_reported);
  ```
  - Notifies auto-tuner of detection capability
  - Enables **error elision**

- ```
  curCD->RequireErrorProbability(
      error_type, num_errors,
      probability,detect_or_fail_over);
  ```
  - Auto- add redundancy to meet requested level of reliability

- ```
  curCD->GetErrorProbability(
                   error_type, num_errors);
  ```
  - Customize action
  ```

Analogues to approximate computing research

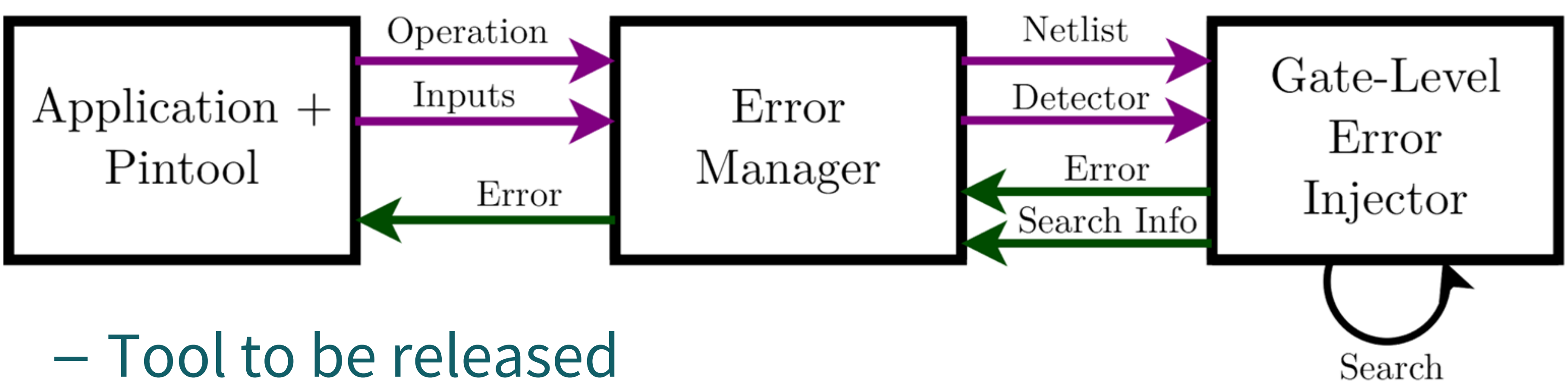
- Compiler techniques for approximate computing
- Propagate loss of accuracy
- Propagate loss of reliability

Debug, test, and tools

- Error and failure injection
- Planned integration with low-level injection
- CD profiler, viz, models, and initial tuner in place

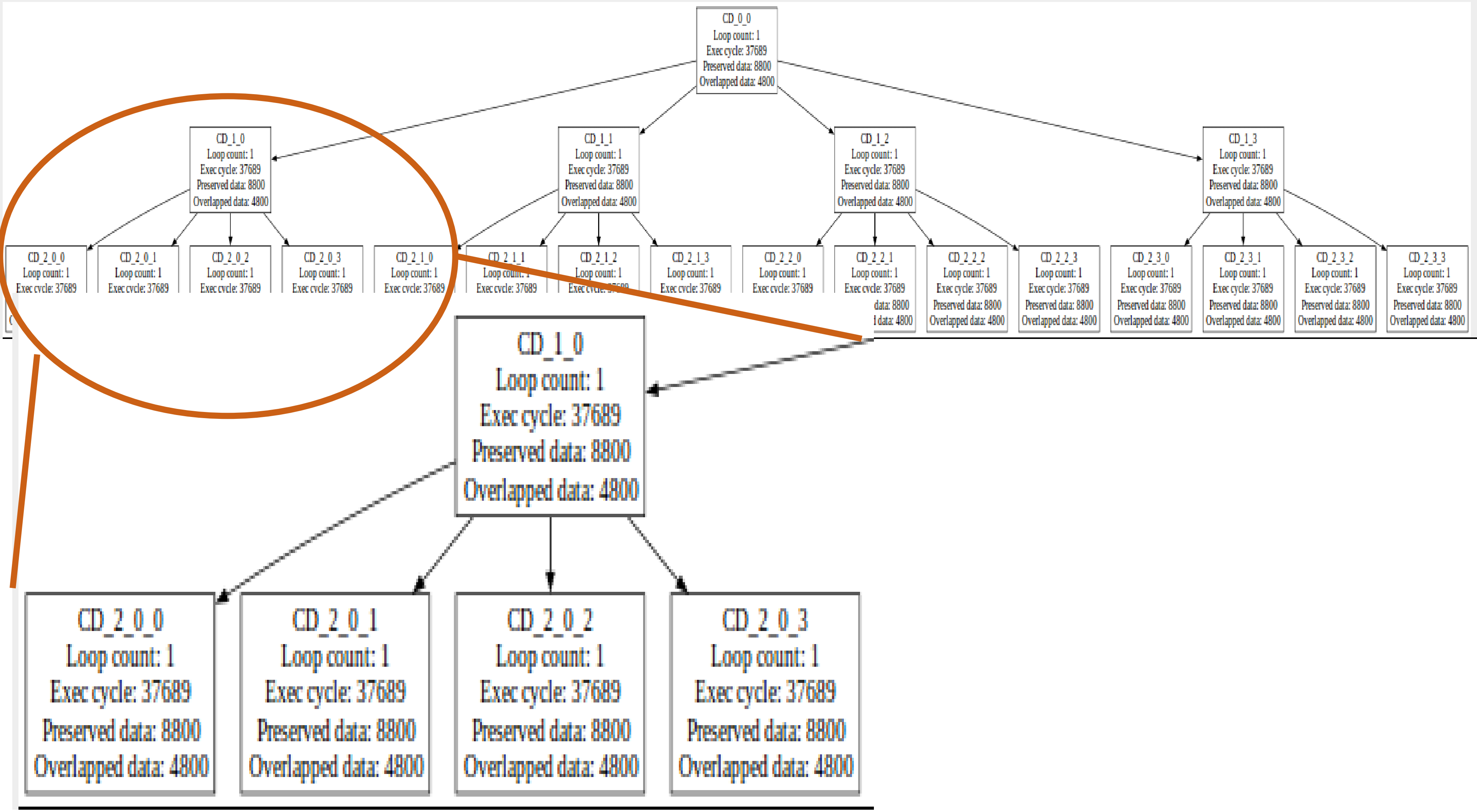
Quick(ish) way to search the error space

- Multi-mode simulation
- Skip over detectable errors

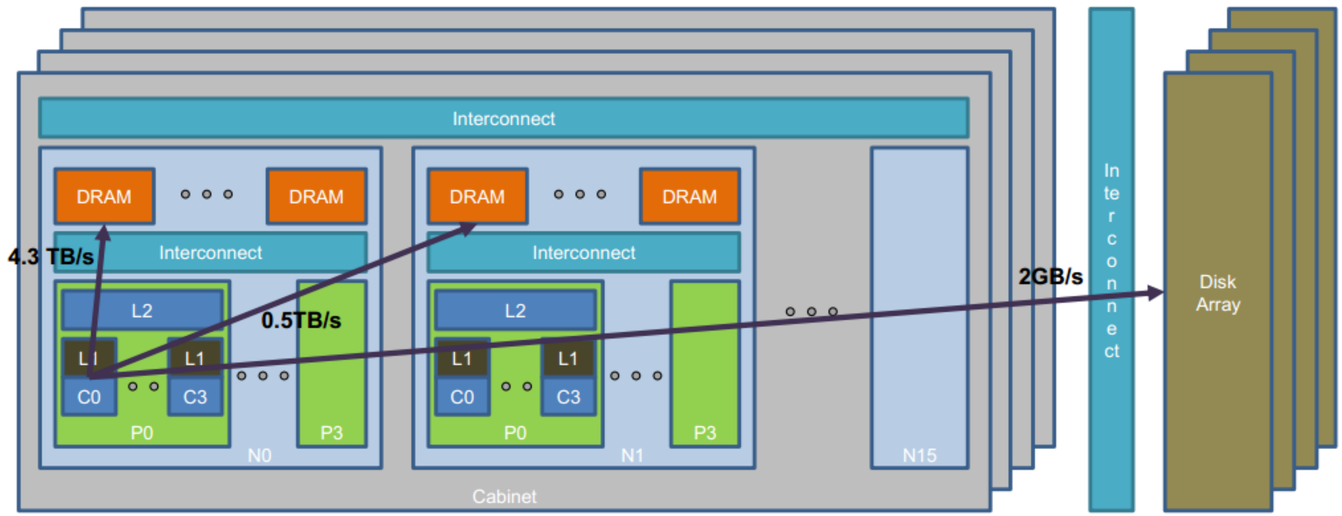


- Tool to be released
  - Uses only public tools

CD Graph corresponding to profile outputcd_profile_2_1_2.conf [# of level = 3]



Machine and error models



[Aggregate BW for 1024 nodes]
- L0 Local DRAM: 483*9GB/sec
- L1 Remote DRAM: 483GB/sec
- L3 Disk: 2.1GB/sec

Component	"Performance"	Error	Error Scaling
Core	10GFLOP/core	Soft error	∝ #cores
Memory	1GB/core	ECC fail	∝ #DRAM chips
Socket	200GB/s /socket	Hard/OS crash	∝ #sockets
System	Hierarchical network	Power module or network	∝ #modules and #cabinets

Input 1: machine configuration

- Physical and storage hierarchies (capacity and BW)
- Error/failure rates at each level of hierarchy
- Simple power model

Input 2: application description

- CD tree, including loops of CDs
- Preservation volumes and possible method
- Overlap of preservation and detection with parent
- Execution time estimate

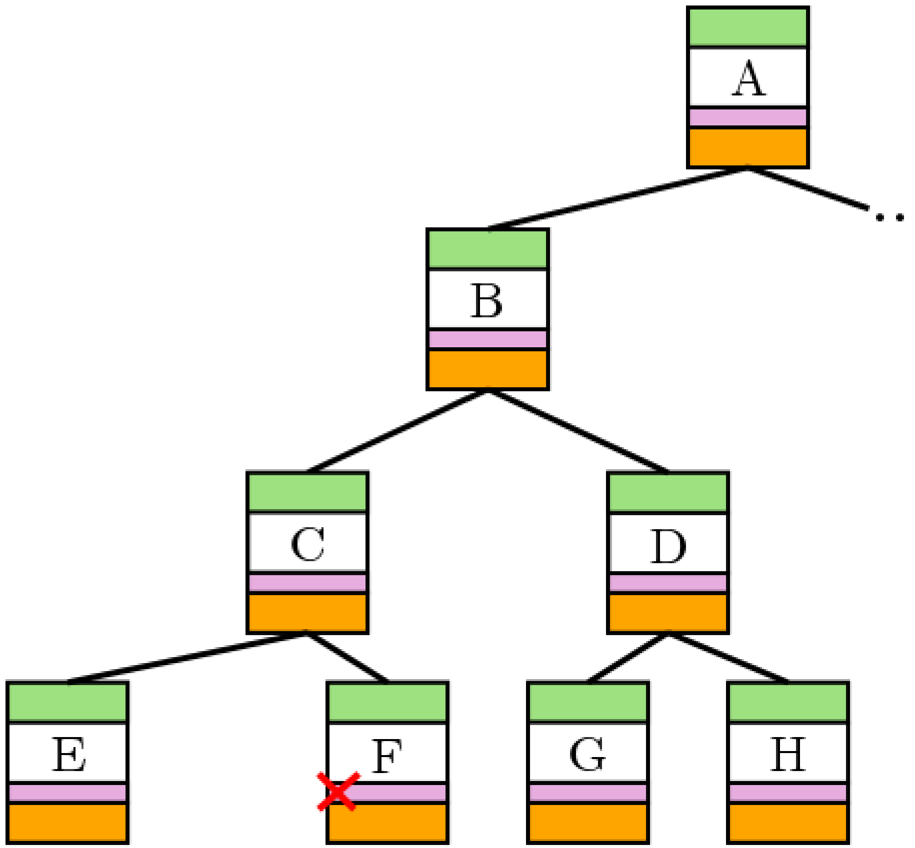
Analytic model for CD behavior

- Overheads from preservation, detection, and recovery

Output efficiency

- Performance, energy, memory

Error Failure Recovery



Analytic Model

Leverage hierarchy and CD semantics

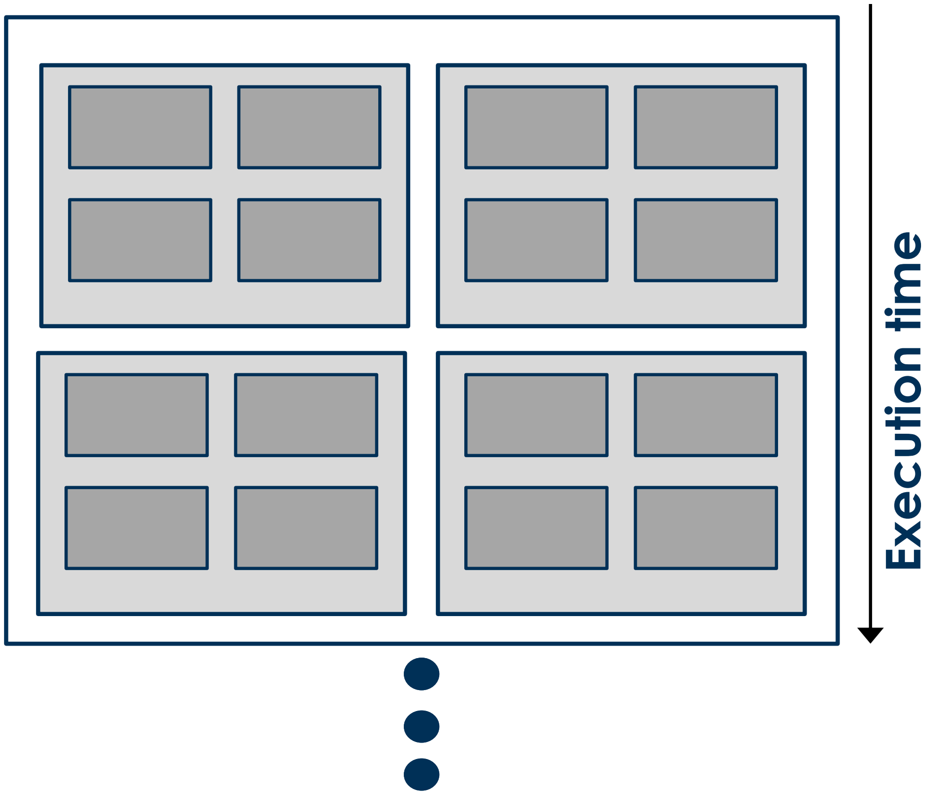
- Uncoordinated "local" actions
- Solve in → out

Application abstracted to CDs

- CD tree
- Volumes of preservation, computation, and communication
- Preservation and recovery options per CD

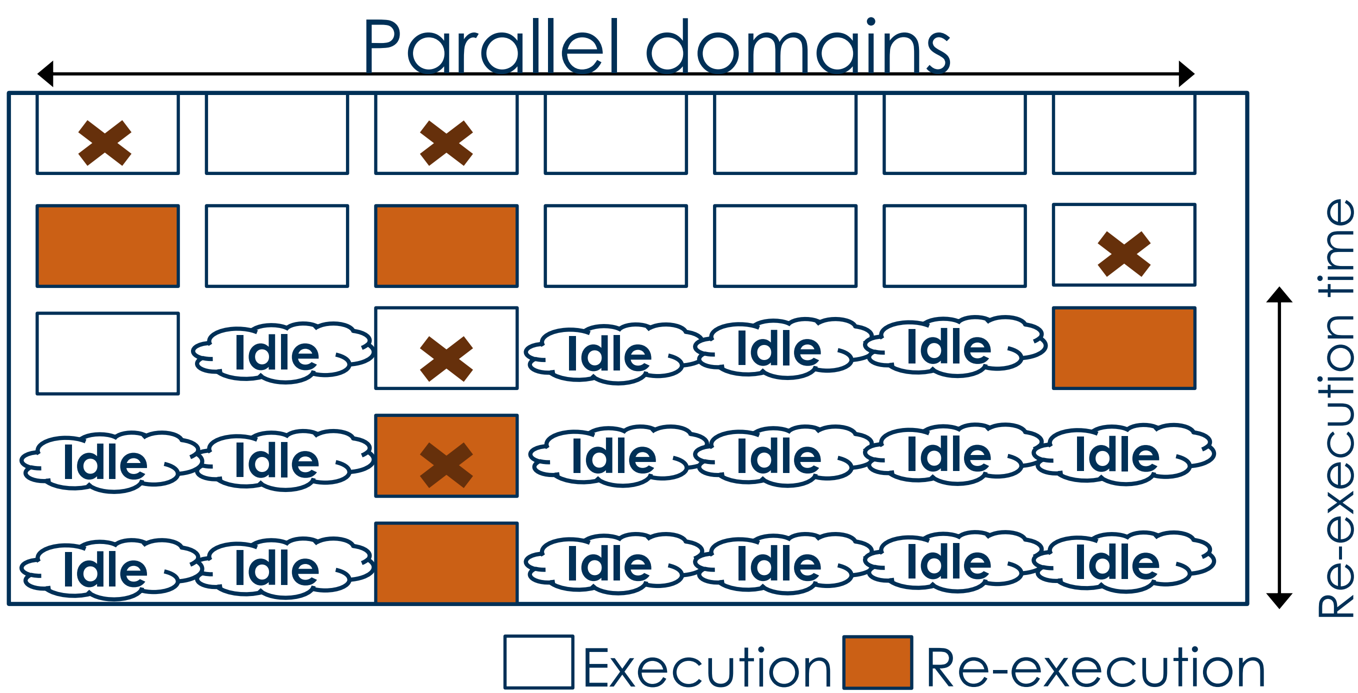
Machine model

- Storage hierarchy
- Communication hierarchy
- Bandwidths and capacities
- Error processes and rates



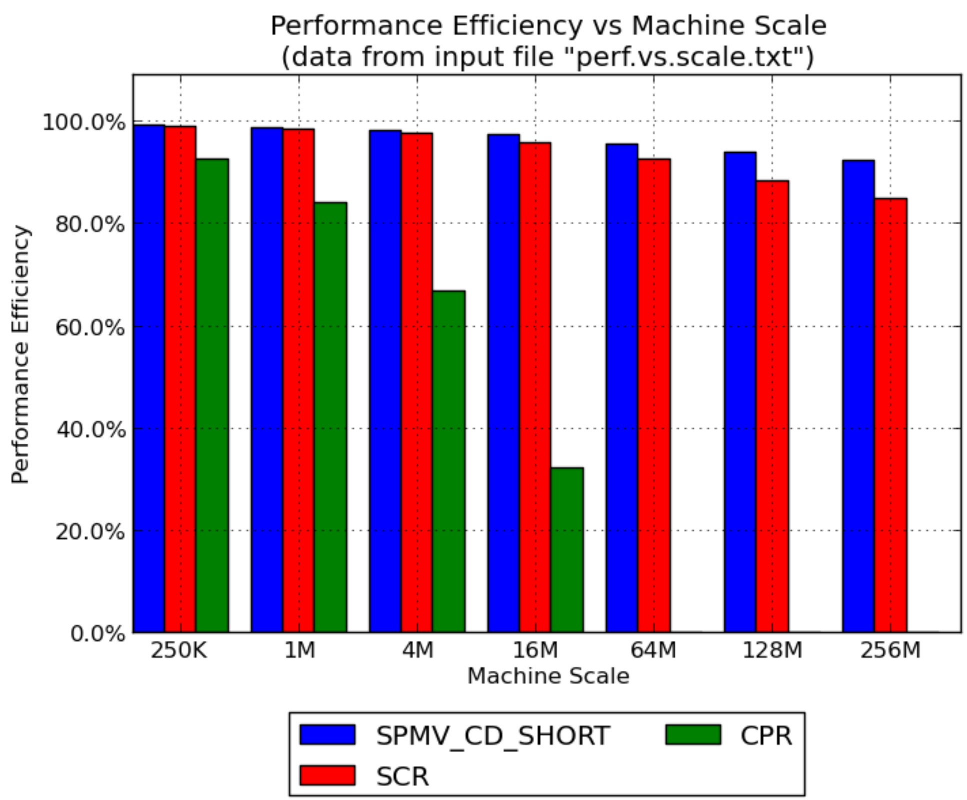
Power model

- CDs that are not re-executing may remain idle
- Actively executing a CD has a relative power of 1
- A node that is idling consumes a relative power of α
  - In our experiments $\alpha = 0.25$



SPMD-oriented analytical model and tuner

- Extrapolated profile
- Machine characteristics
- Tuning space and models



Auto-tuned cross-layer resilience!

– Iterate with error injection
– Intelligent search exploration

Execution model progress

- Building systems is hard and tricky
- Limited release of single-node runtime
- MPI runtime very close
  - Lots of distributed programming issues
  - Lots of current sad state of FT issues
- Open source soon on Bitbucket
  - Initially only for soft errors

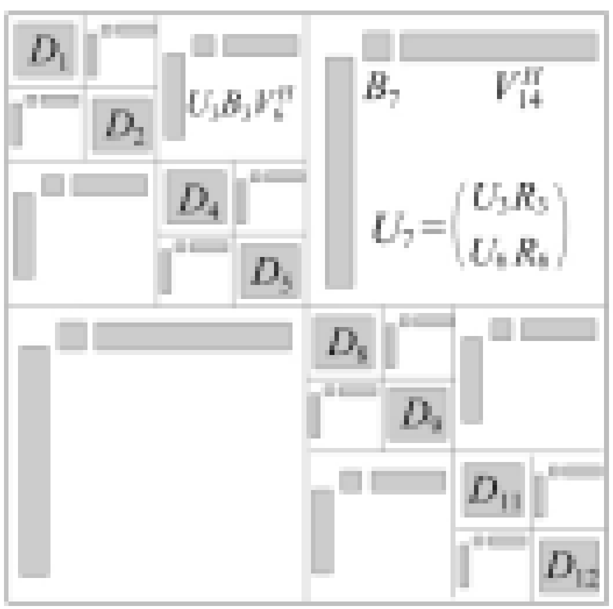
Already useful and collaborations in progress

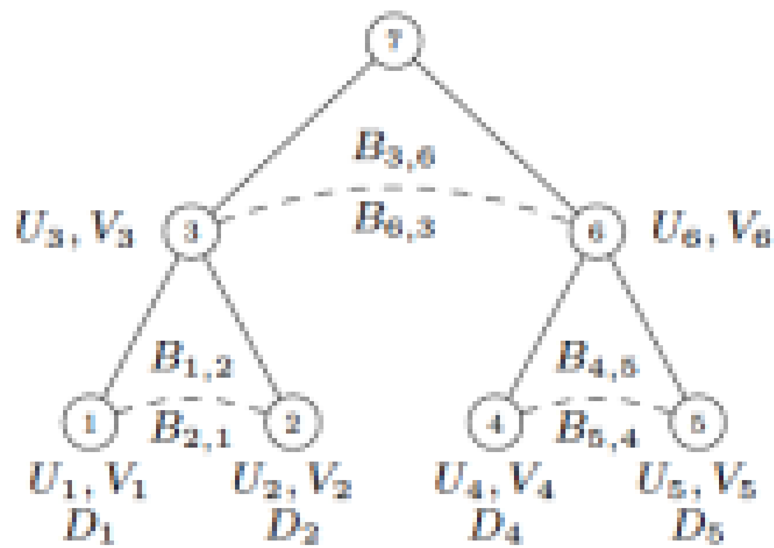
- Reaching down to hardware in FF2
- Global address space with DEGAS
- Task-based execution in Legion and SWARM
- DSL-facing in Stanford's PSAAP II
- Algorithmic approach within TOORSES

TOORSES fault-tolerant hierarchical solver

– Brian Austin, Eric Roman, and Xiaoye (Sherry) Li LBNL

– Hierarchical semi-separable representation





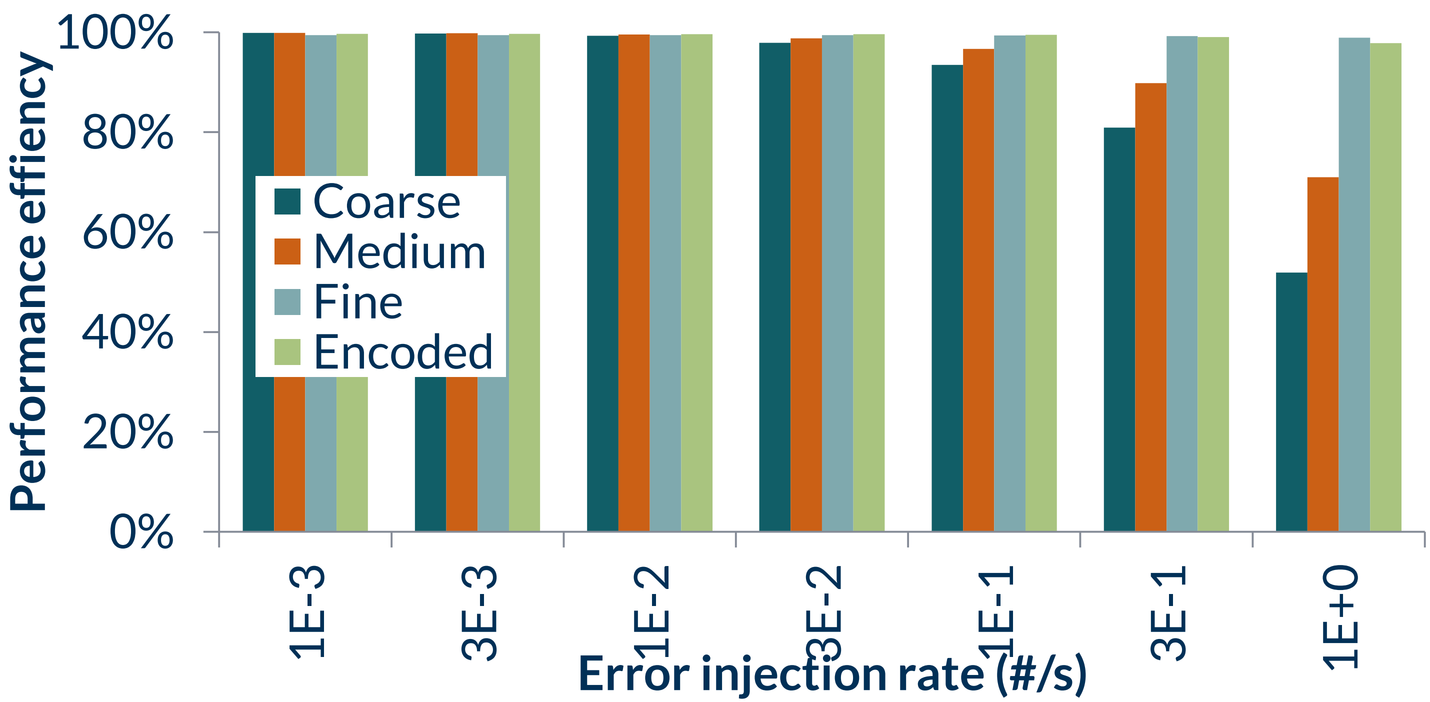
Add CDs at different granularities

– Hierarchical and partial preservation

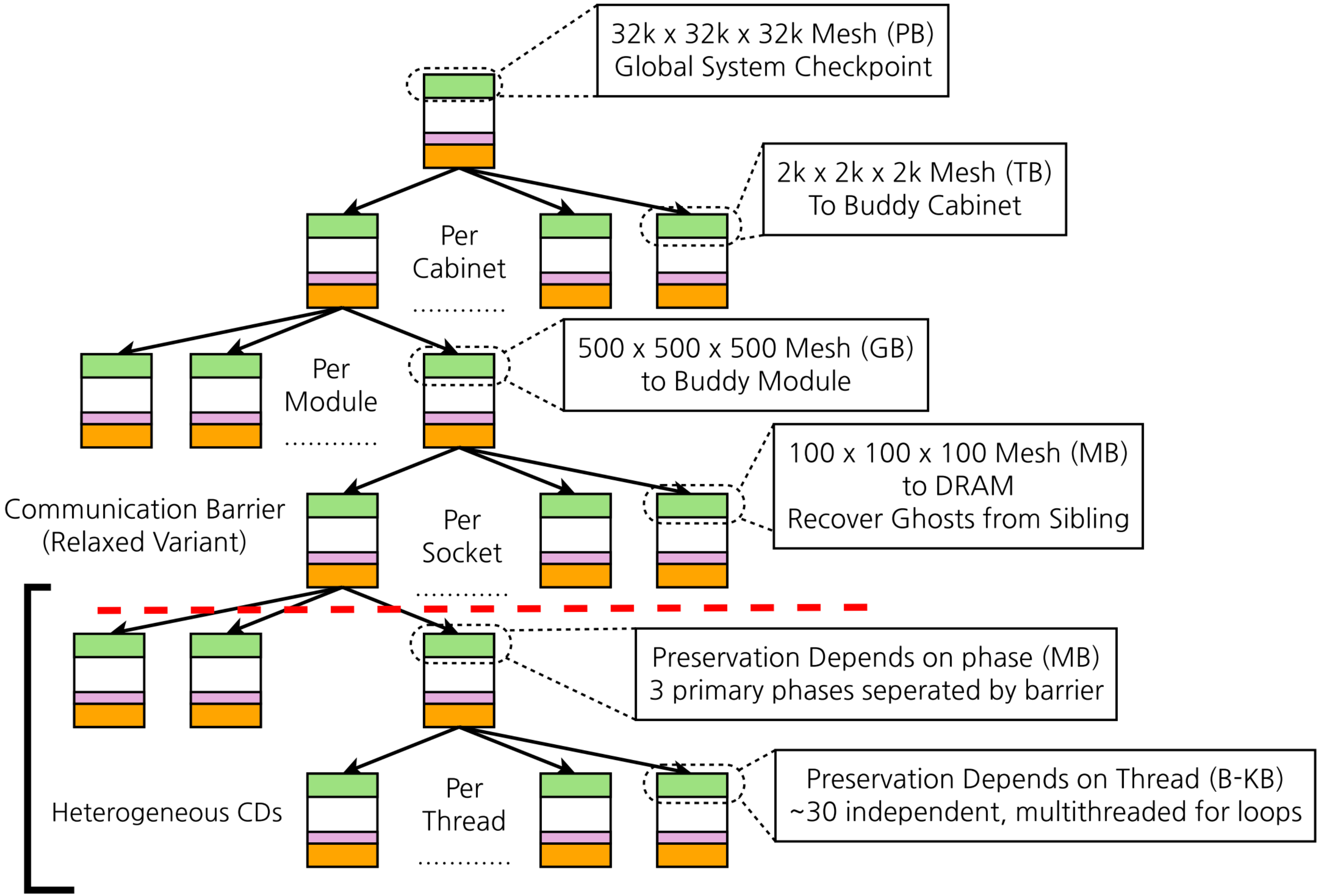
Add algorithmic and cheap detection

Compare to:

– Algorithmic recovery with redundant computation

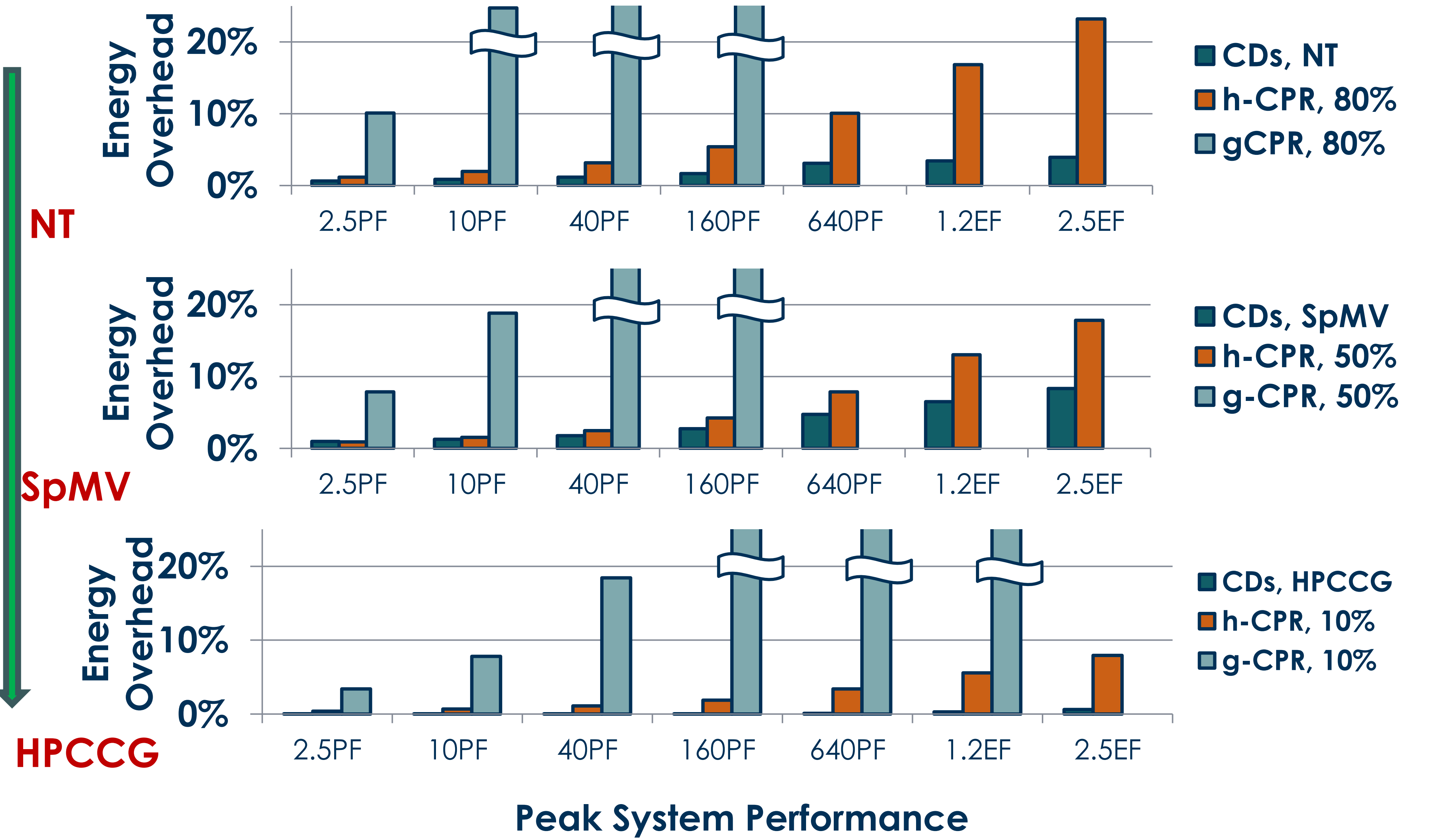


LULESH CD mapping example

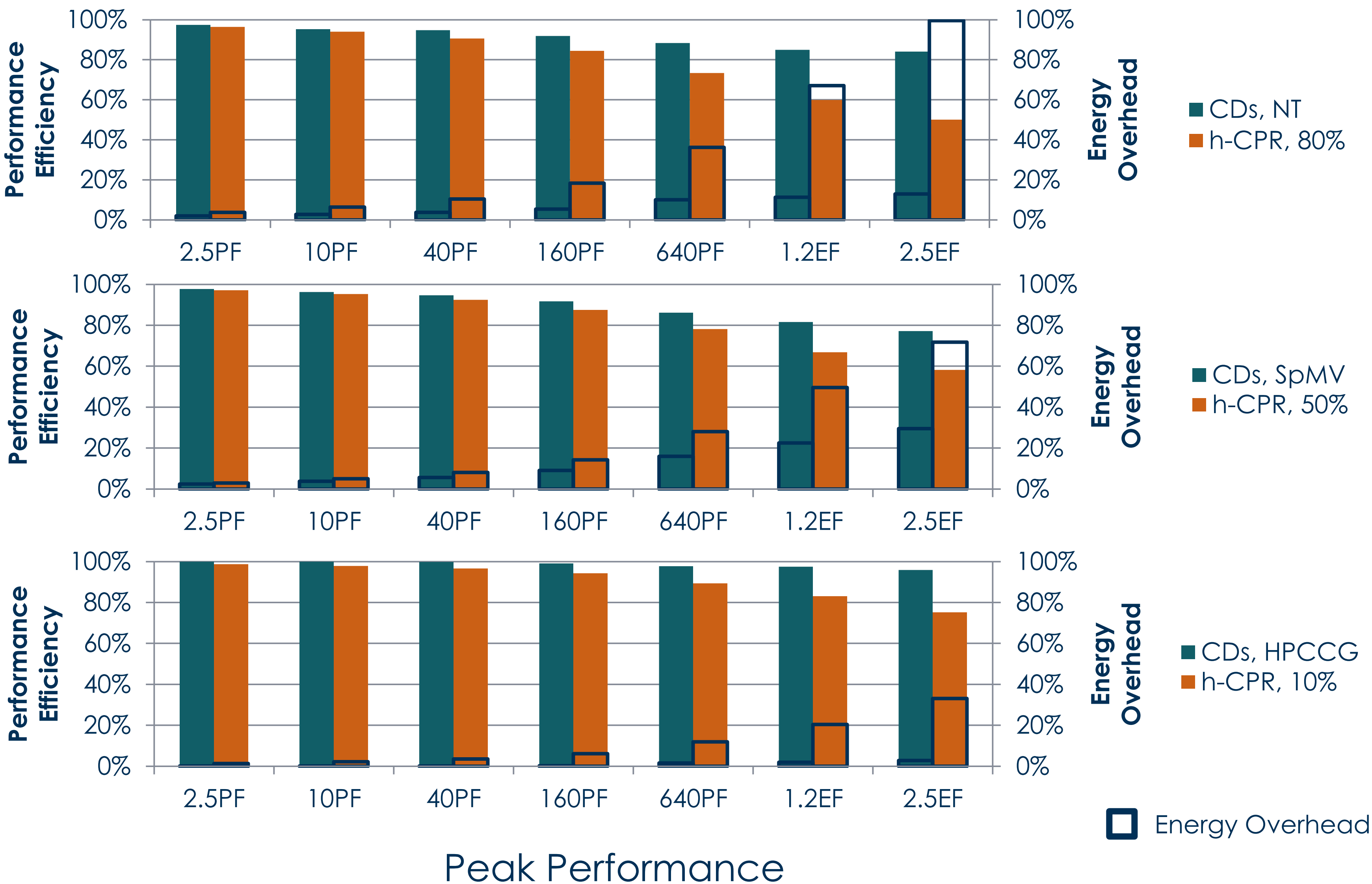


Autotuned CDs perform well

CDs improve energy efficiency at scale



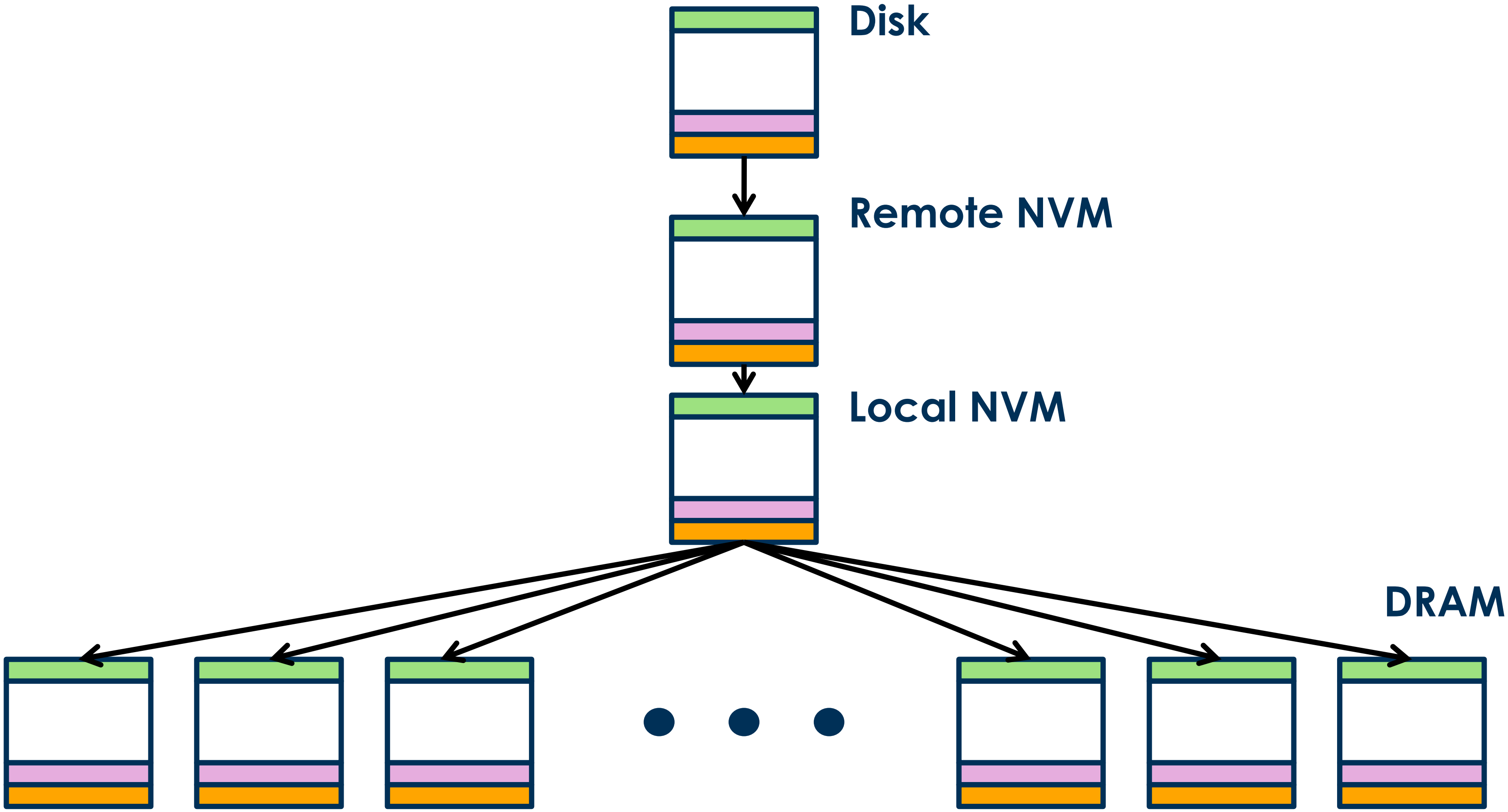
10X failure rate emphasizes CD benefits



What if my application has many barriers?

– Can’t really form a tree?

SPMV:
local recovery and partial preservation



Partial preservation via sibling or parent where appropriate

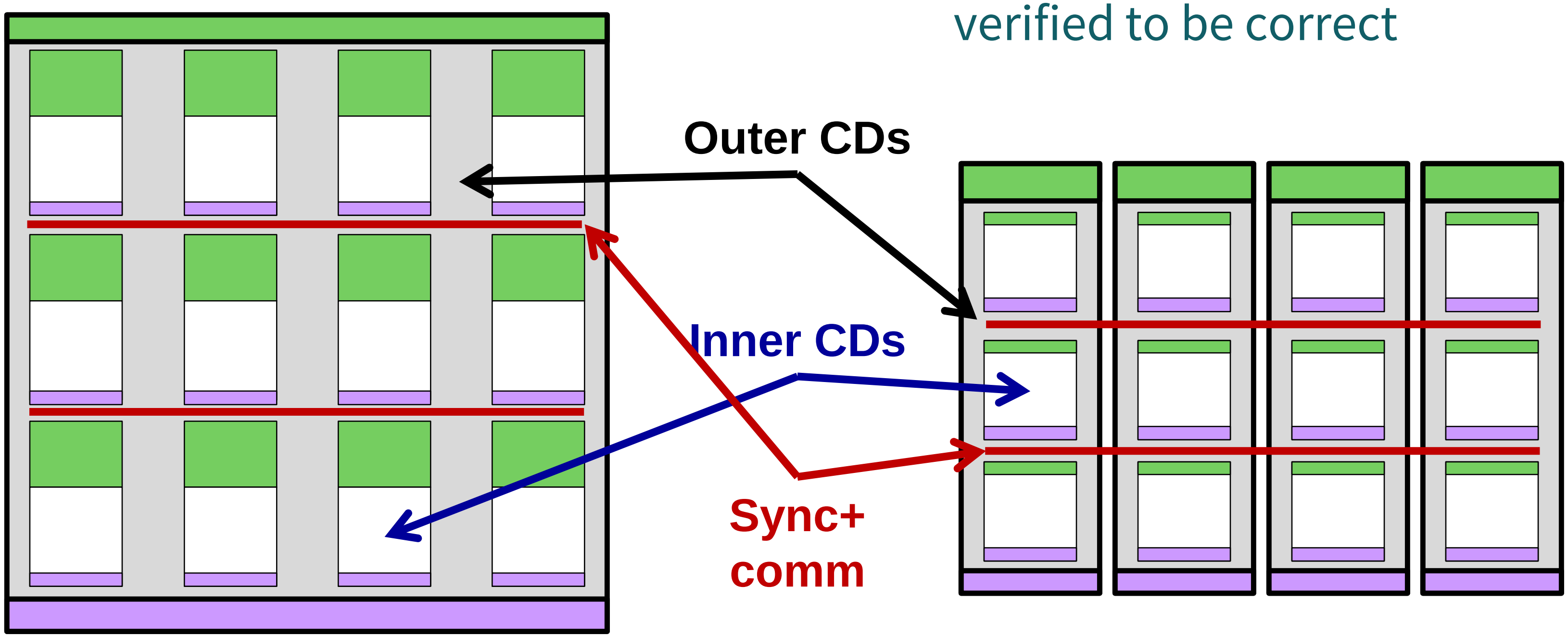
Inter-CD communication?

Strict CDs do not communicate

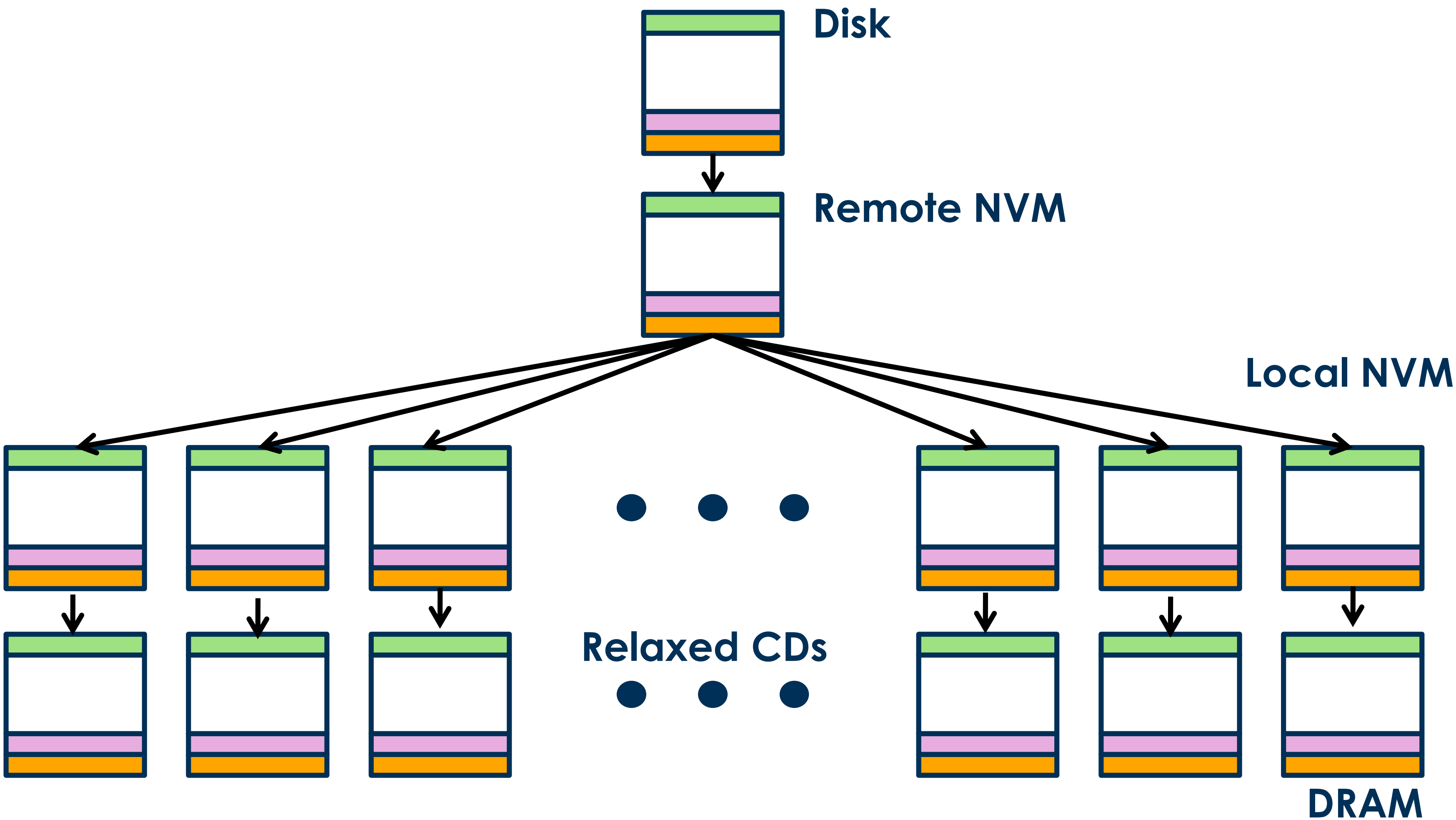
– Only communicate when in same CD context
– Overheads for strict containment can be high

Relaxed CDs enable inter-CD communication

– Maintain CD semantics w/ uncoordinated recovery
– Some data "preserved" via logging
– All communicated data still verified to be correct

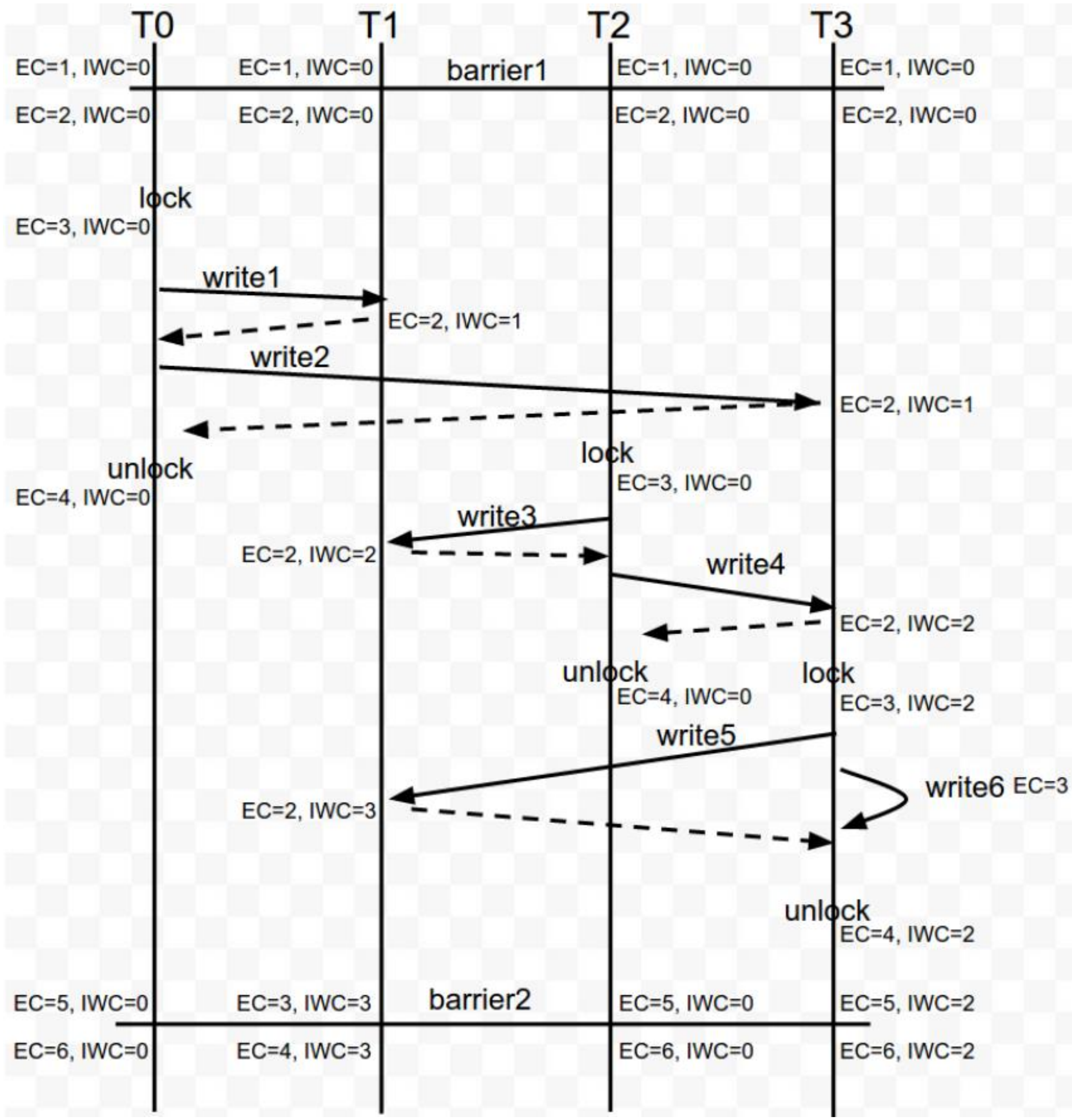


SPMV:
local recovery and partial preservation



Partial preservation via sibling or parent where appropriate

Fun with logging protocols

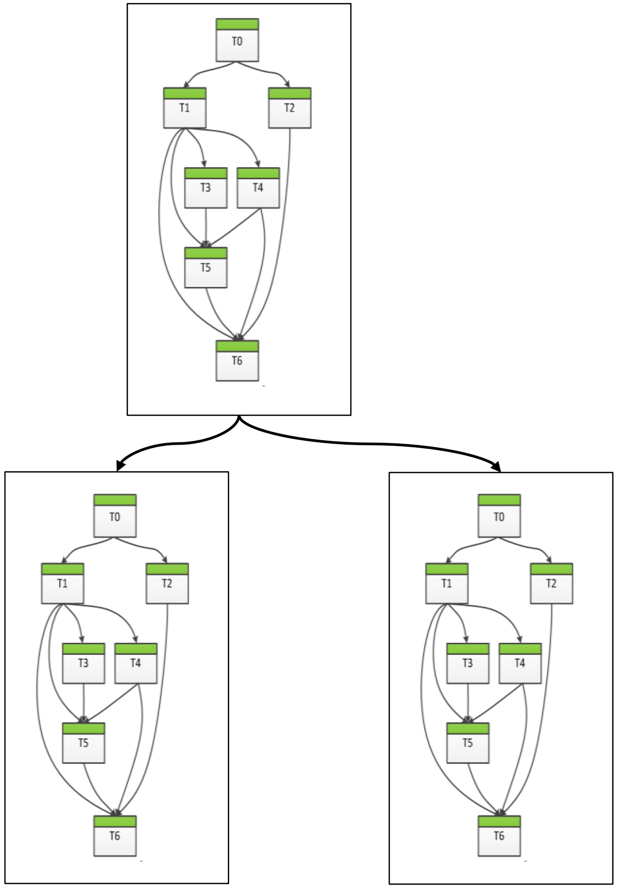


What about tasks?

- CDs are great natural fit
  - CDs + Legion
    - Stanford project led by Alex Aiken
  - CDs + Swarm
    - Spinoff from UDel led by Guang Gao
  - Perhaps also with *SS / Nachos
    - Barcelona Supercomputing Centers

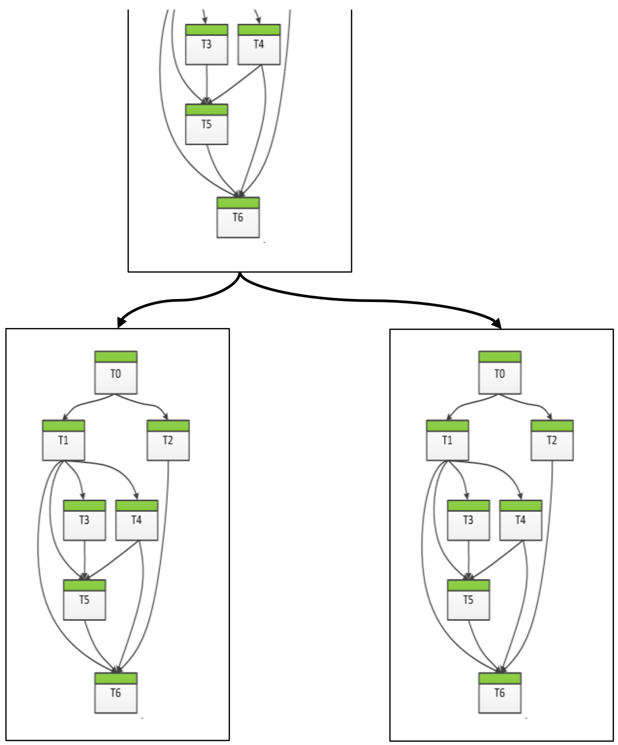
Legion resilience

- Propagate failures up the dependence chain
- Utilize region copies to minimize reexecutions



Legion + CDs resilience

- Model-guided management of copies
- Optimized reexecution propagation stop points
- Detection and specification semantics
- Integration with other resilience mechanisms

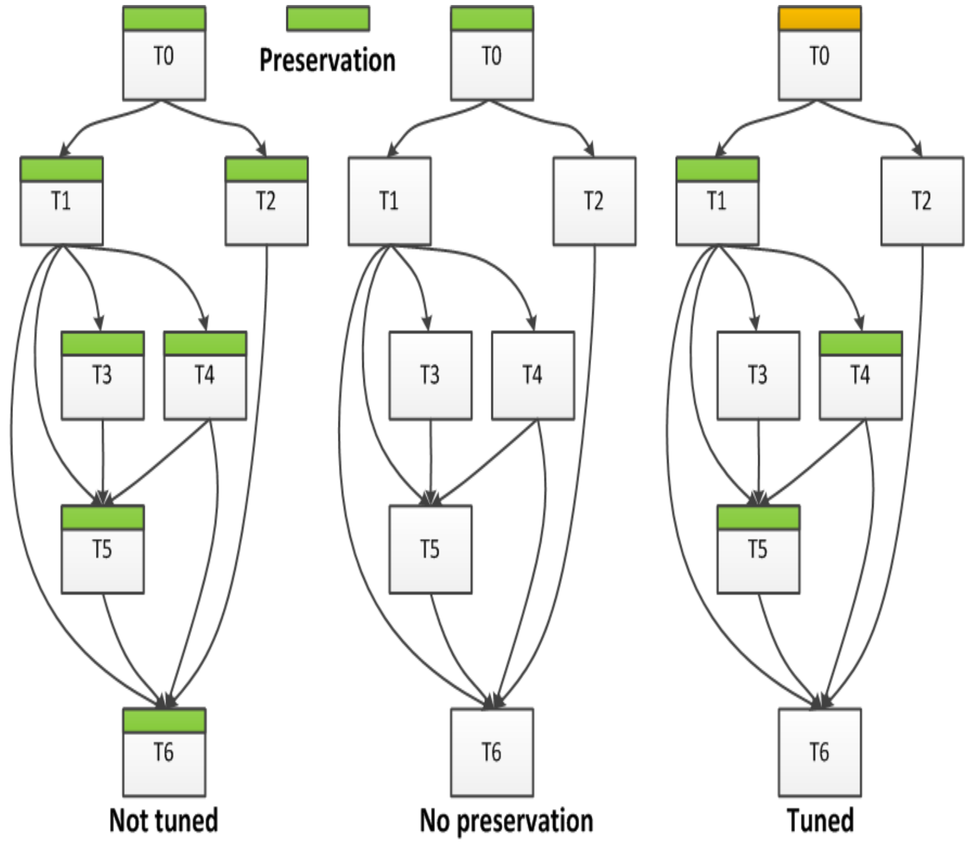


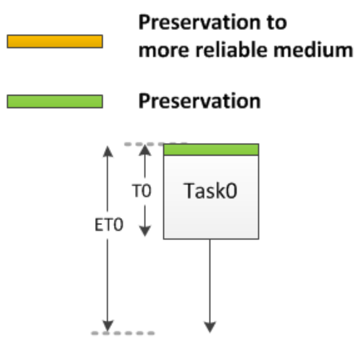
Use Legion copies for CD preservation

Optimize for efficiency

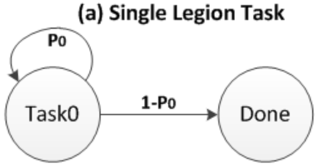
- When to add copies
- Where to put copies to survive failures
- When to free copies

Account for different failure modes and rates





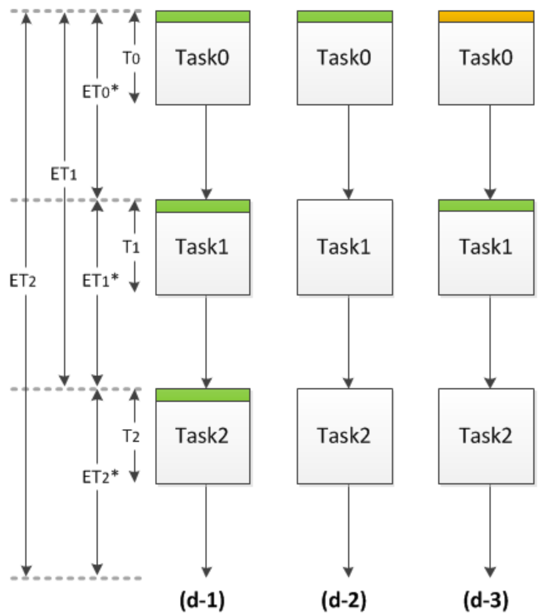
(a) Single Legion Task



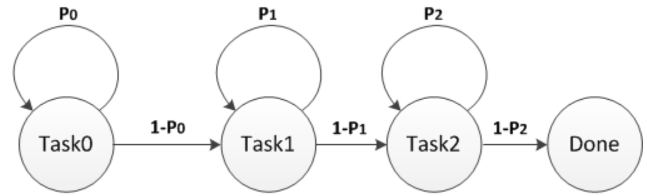
(b) Markov chain model of (a)

$$ET_0 = \sum_{i=0}^{\infty} P_0^i (1 - P_0) \times (T_0 + T_{0,r}) \times (i + 1)$$

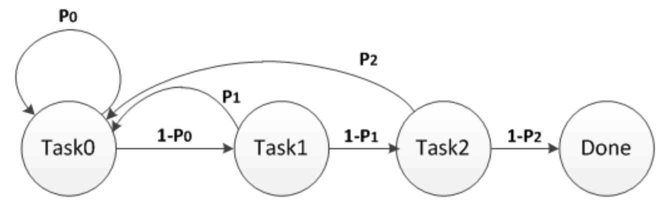
(c) Expected execution time of Task1



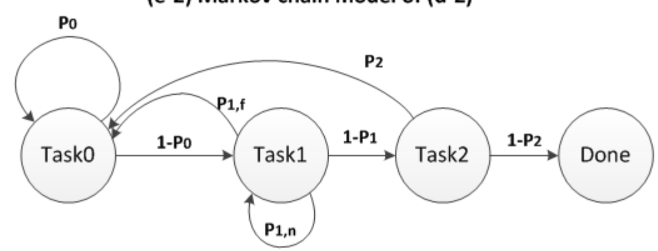
(d) Sequential tasks



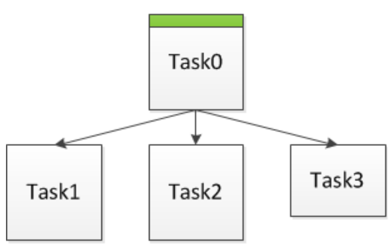
(e-1) Markov chain model of (d-1)



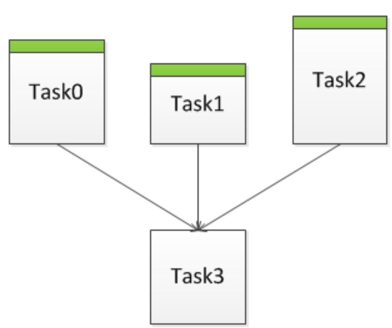
(e-2) Markov chain model of (d-2)



(e-3) Markov chain model of (d-3)



(f) three-successor Legion Tasks



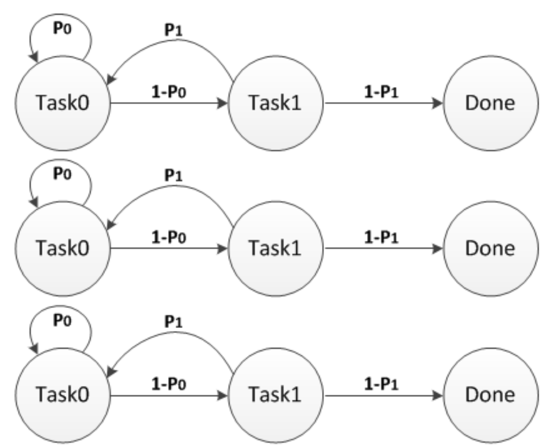
(g) three-predecessor Legion Task

$$ET_1 = ET_0 + \sum_{i=0}^{\infty} P_1^i (1 - P_1) \times ((T_1 + ET_0) \times i + T_1)$$

$$ET_2 = ET_0 + \sum_{i=0}^{\infty} P_2^i (1 - P_2) \times ((T_2 + ET_0) \times i + T_2)$$

$$ET_3 = ET_0 + \sum_{i=0}^{\infty} P_3^i (1 - P_3) \times ((T_3 + ET_0) \times i + T_3)$$

$$ET_3 = \max(ET_0, ET_1, ET_2) + \sum_{i=0}^{\infty} P_3^i (1 - P_3) \times \{(T_3 + \max(ET_0^*, ET_1^*, ET_2^*)) \times i + T_3\}$$



(h) Markov chain model of (f) and (g)

Assumption/fear: reliability bounds performance

- Errors may corrupt results and failures kill applications

What is the error rate?

- Like today: keep ignoring the problem
- Much higher: **need detection and recovery**
- CDs abstract, scalable, and tunable

What is the failure rate?

- Like today: hierarchical checkpoint restart
- Higher: **specialize preservation and recovery**
- CDs are portable and tunable

Is it really a problem?

- CDs are **general and analyzable**
- CDs are **composable**?

Conclusion

Containment domains

- **Abstract** constructs for resilience concerns & techniques
- **Proportional** and application/machine tuned resilience
- **Hierarchical & distributed** preservation, and recovery
- **Analyzable** and amendable to automatic optimization
- **Scalable** with high relative energy efficiency
- **Heterogeneous** to match emerging architecture

http://lph.ece.utexas.edu/public/CDs

Thank You!

– Please find the slides at https://lph.ece.utexas.edu/merez/MattanErez/ExacaleResilienceShort0715