Characteristics of *Adaptive* Runtime Systems in HPC

Laxmikant (Sanjay) Kale

http://charm.cs.illinois.edu

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

What runtime are we talking about?

- Java runtime:
  - JVM + Java class library
  - Implements JAVA API
- MPI runtime:
  - Implements MPI standard API
  - Mostly mechanisms
- I want to focus on runtimes that are "smart"
  - i.e. include strategies in addition mechanisms
  - Many mechanisms to enable adaptive strategies

Why?

And what kind of adaptive runtime system I have in mind?

Let us take a detour

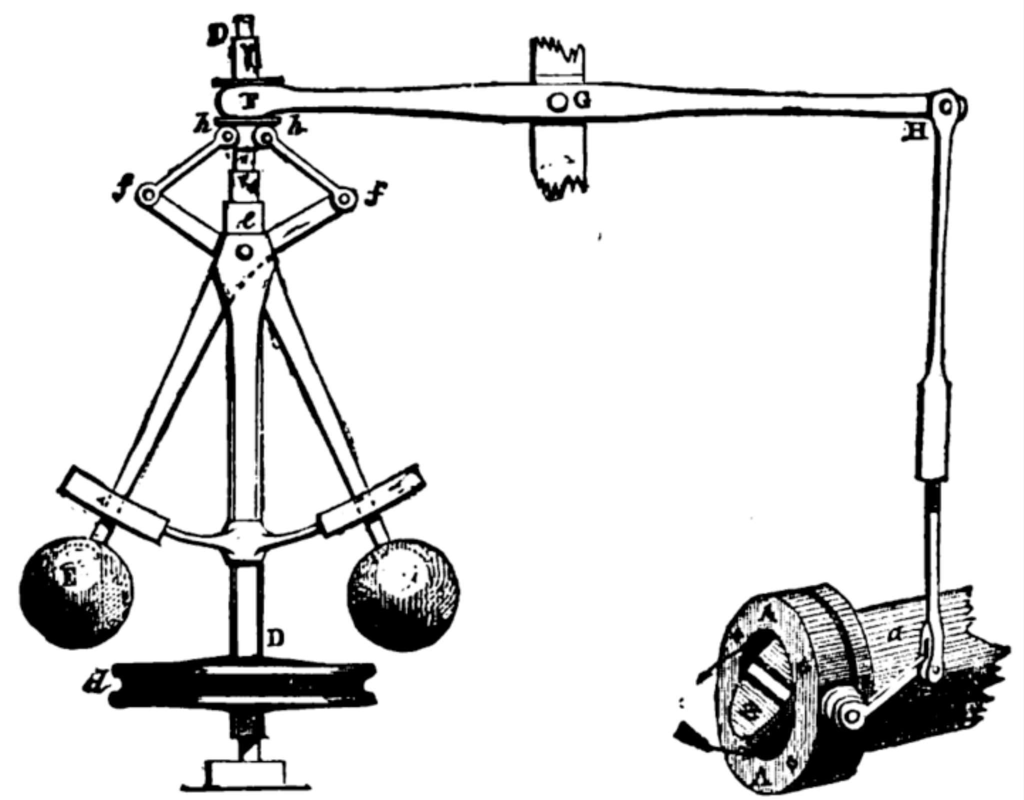
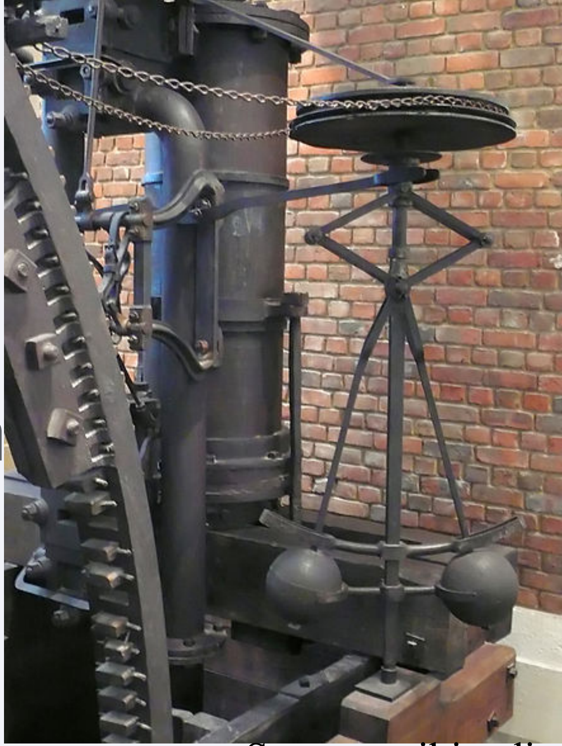


FIG. 4.—*Governor and Throttle-Valve.*

Source: wikipedia

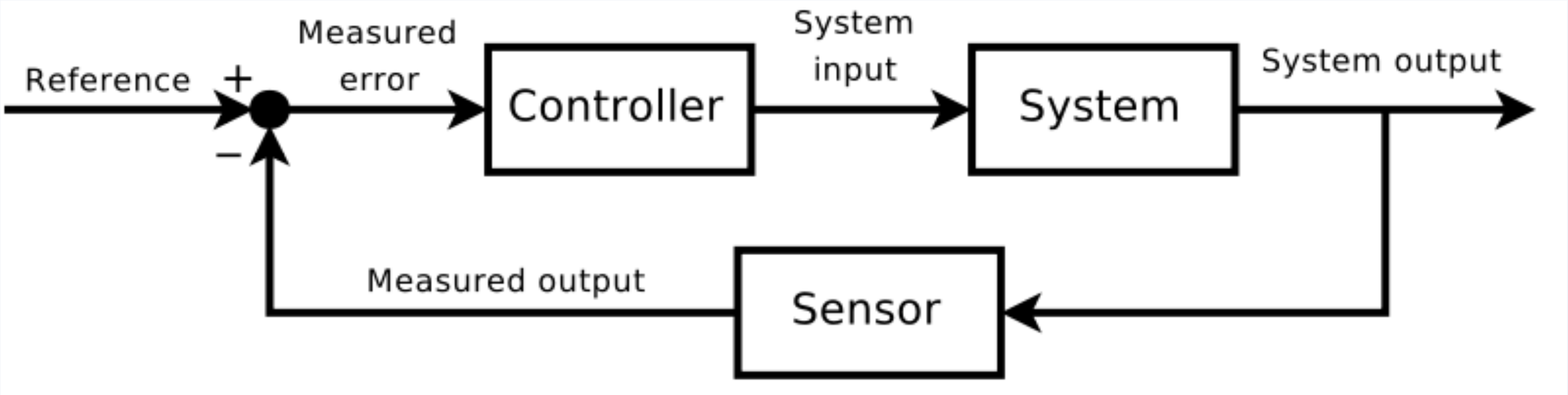
Governors


Source: wikipedia

- Around 1788 AD, James Watt and Mathew Boulton solved a problem with their steam engine
  - They added a cruise control… well, RPM control
  - How to make the motor spin at the same constant speed
  - If it spins faster, the large masses move outwards
  - This moves a throttle valve so less steam is allowed in to push the prime mover

Feedback Control Systems Theory

- This was interesting:
  - You let the system “misbehave”, and use that misbehavior to correct it..
  - Of course, there is a time-lag here
  - Later Maxwell wrote a paper about this, giving impetus to the area of “control theory”

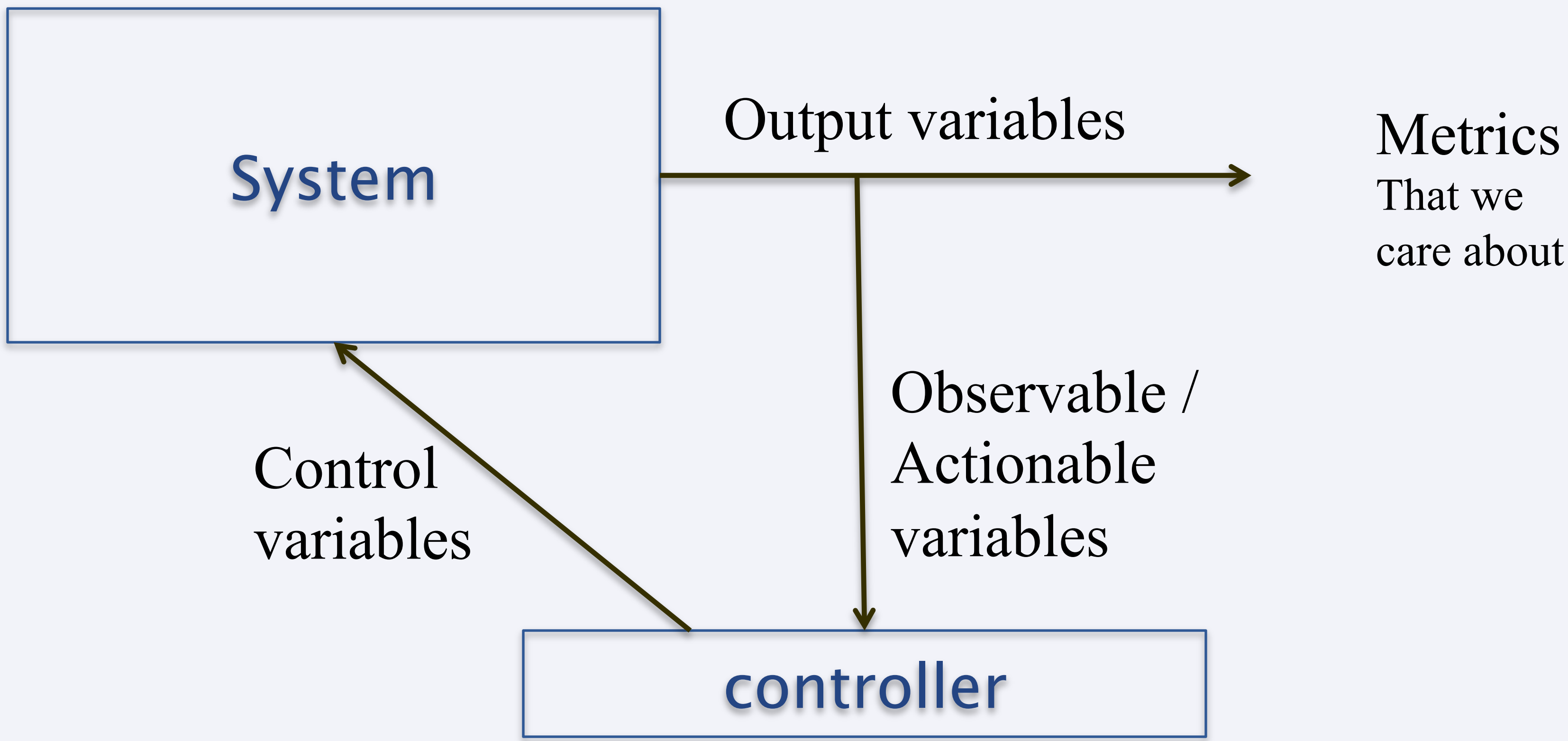


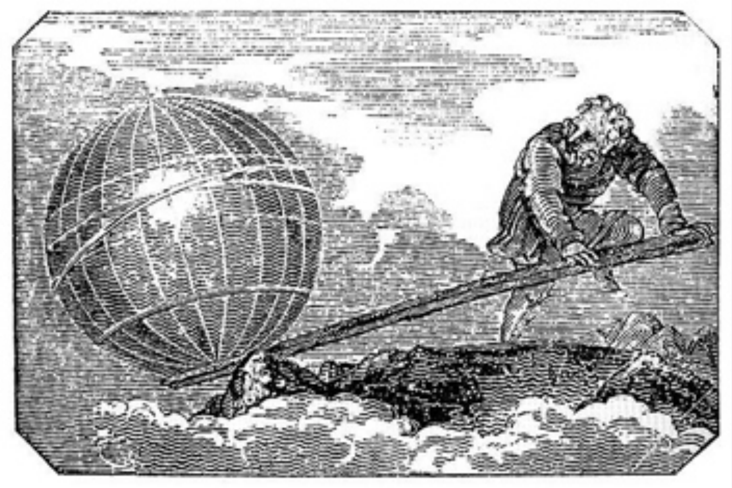
Source: wikipedia

Control theory

- The control theory was concerned with stability, and related issues
  - Fixed delay makes for highly analyzable system with good math demonstration
- We will just take the basic diagram and two related notions:
  - Controllability
  - Observability

A modified system diagram





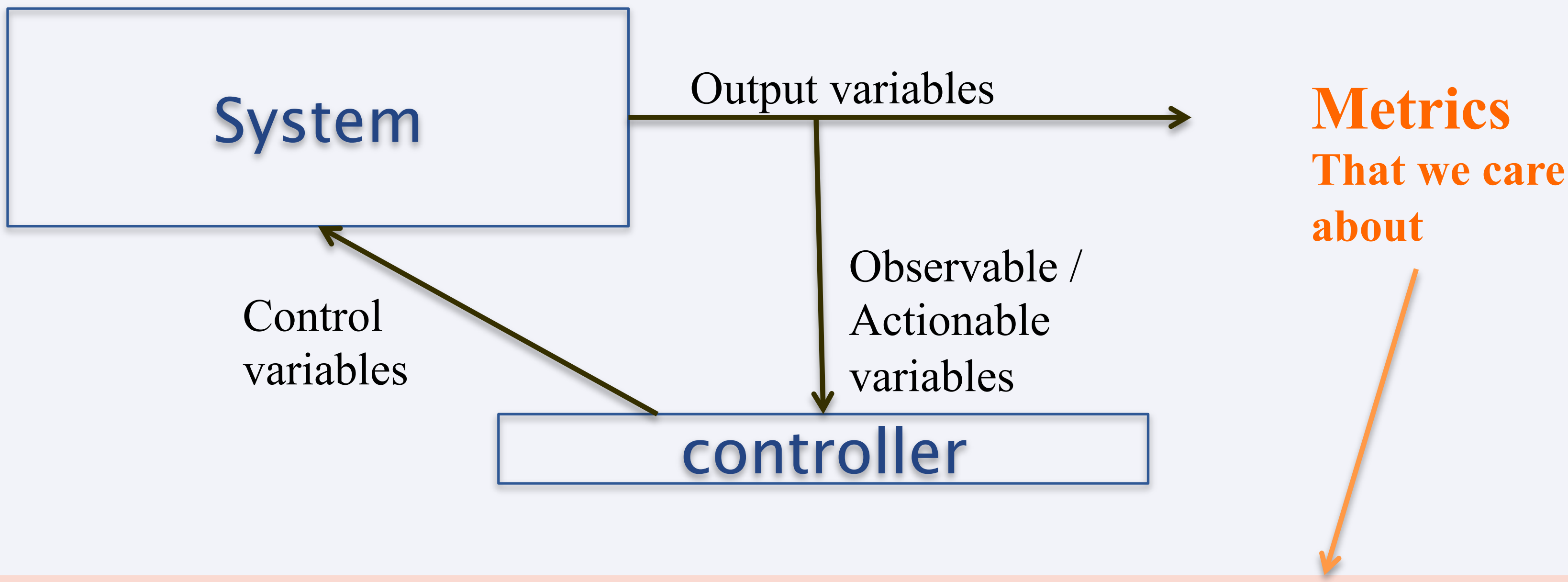
Source: wikipedia

Archimedes is supposed to have said, of the lever: Give me a place to stand on, and I will move the Earth

Need to have the lever

- **Observability:**
  - If we can't observe it, can't act on it
- Controllability:
  - If no appropriate control variable is available, we can't control the system
    - (bending the definition a bit)
- So: <u>an effective control system needs to have a rich set of observable and controllable variables</u>

A modified system diagram



These include one or more:

- Objective functions (minimize, maximize, optimize)
- Constraints: “must be less than”, ..

Feedback Control Systems in HPC?

- Let us consider two “systems”
  - And examine them for opportunities for feedback control
- A parallel “job”
  - A single application running in some partition
- A parallel machine
  - Running multiple jobs from a queue

A Single Job

- System output variables that we care about:
  - (Other than the job's science output)
  - Execution time, energy, power, memory usage, ..
  - First two are objective functions
  - Next two are (typically) constraints
  - We will talk about other variables as well, later
- What are the observables?
  - Maybe message sizes, rates? Communication graphs?
- What are the control variables?
  - Very few…. Maybe MPI buffer size? bigpages?

Control System for a single job?

- Hard to do, mainly because of the paucity of control variables
- This was a problem with “Autopilot”, Dan Reed’s otherwise exemplary research project
  - Sensors, actuators and controllers could be defined, but the underlying system did not present opportunities
- We need to “open up” the single job to expose more controllable knobs

Alternatives

- Each job has its own ARTS control system, for sure
- But should this be:
  - Specially written for that application?
  - A common code base?
  - A framework or DSL that includes an ARTS?
- This is an open question, I think..
  - But it must be capable of interacting with the machine-level control system
- My opinion:
  - Common RTS, but specializable for each application

The Whole Parallel Machine

- Consists of nodes, job scheduler, resource allocator, job queue, ..
- Output variables:
  - Throughput, Energy bill, energy per unit of work, power, availability, reliability, ..
- Again, very little control
  - About the only decision we make is which job to run next, and which nodes to give to it..

The Big Question/s:

How to add more control variables?

How to add more observables?

One method we have explored

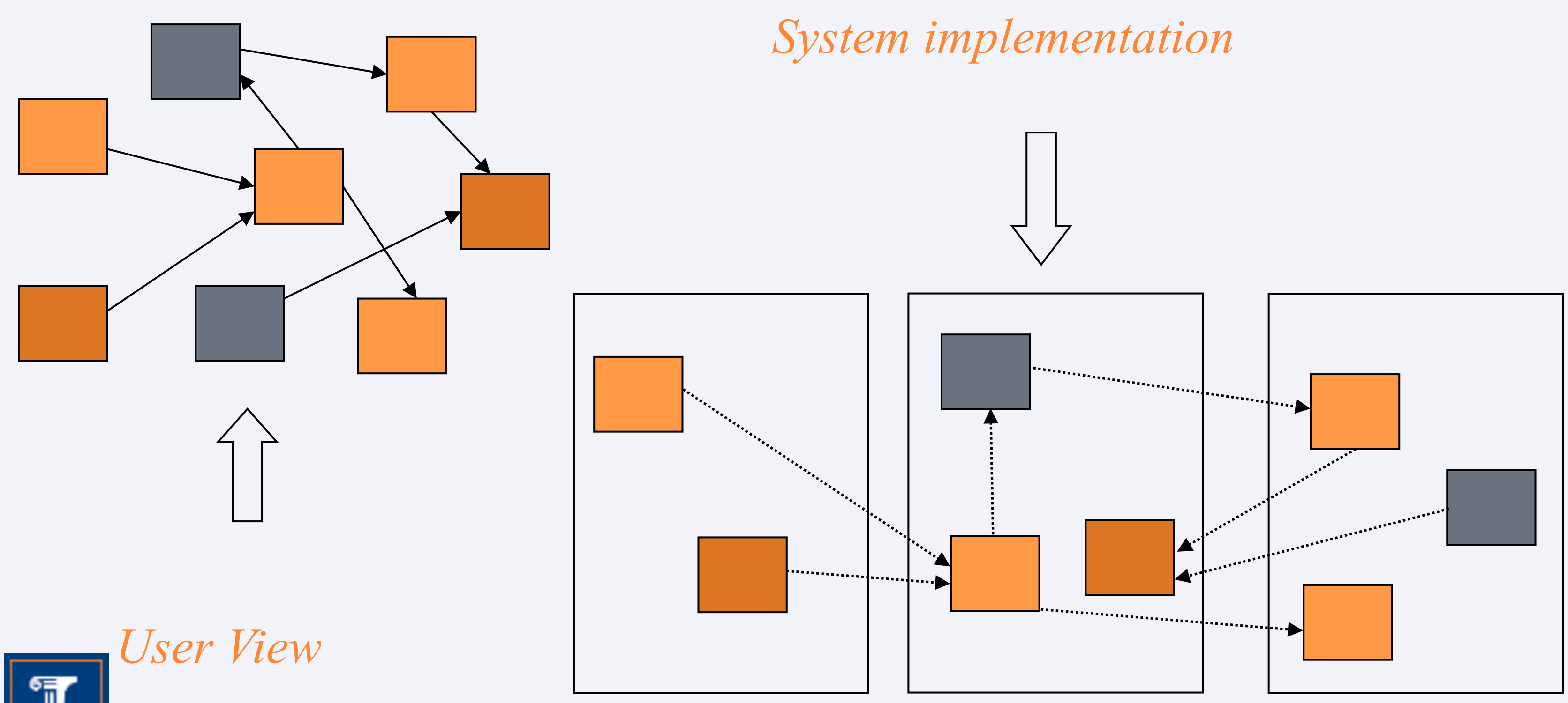
- Overdecomposition and processor independent programming

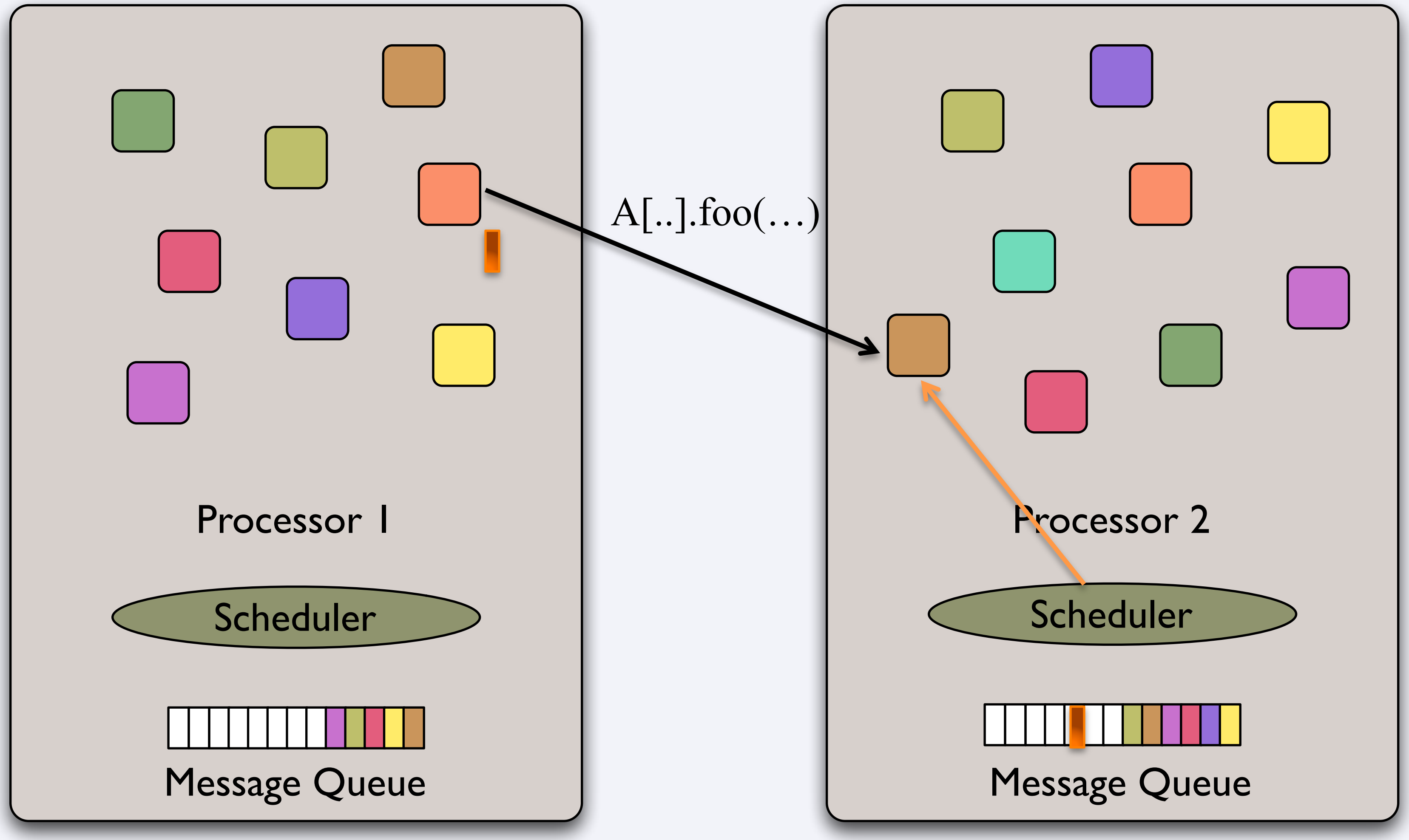
Object based over-decomposition

- Let the programmer decompose computation into objects
  - Work units, data-units, composites
- Let an intelligent runtime system assign objects to processors
  - RTS can change this assignment during execution
- This empowers the control system
  - A large number of observables
  - Many control variables created

Object-based over-decomposition: Charm++

- Multiple "indexed collections" of C++ objects
- Indices can be multi-dimensional and/or sparse
- Programmer expresses communication between objects
  - with no reference to processors




A[..].foo(…)
Processor 1
Scheduler
Message Queue
Processor 2
Scheduler
Message Queue

Note the control points created

- Scheduling (sequencing) of multiple method invocations waiting in scheduler's queue
- Observed variables: execution time, object communication graph (who talks to whom)
- Migration of objects
  - System can move them to different processors at will, because..
- This is already very rich…
  - What can we do with that??

Optimizations Enabled/Enhanced by These New Control Variables

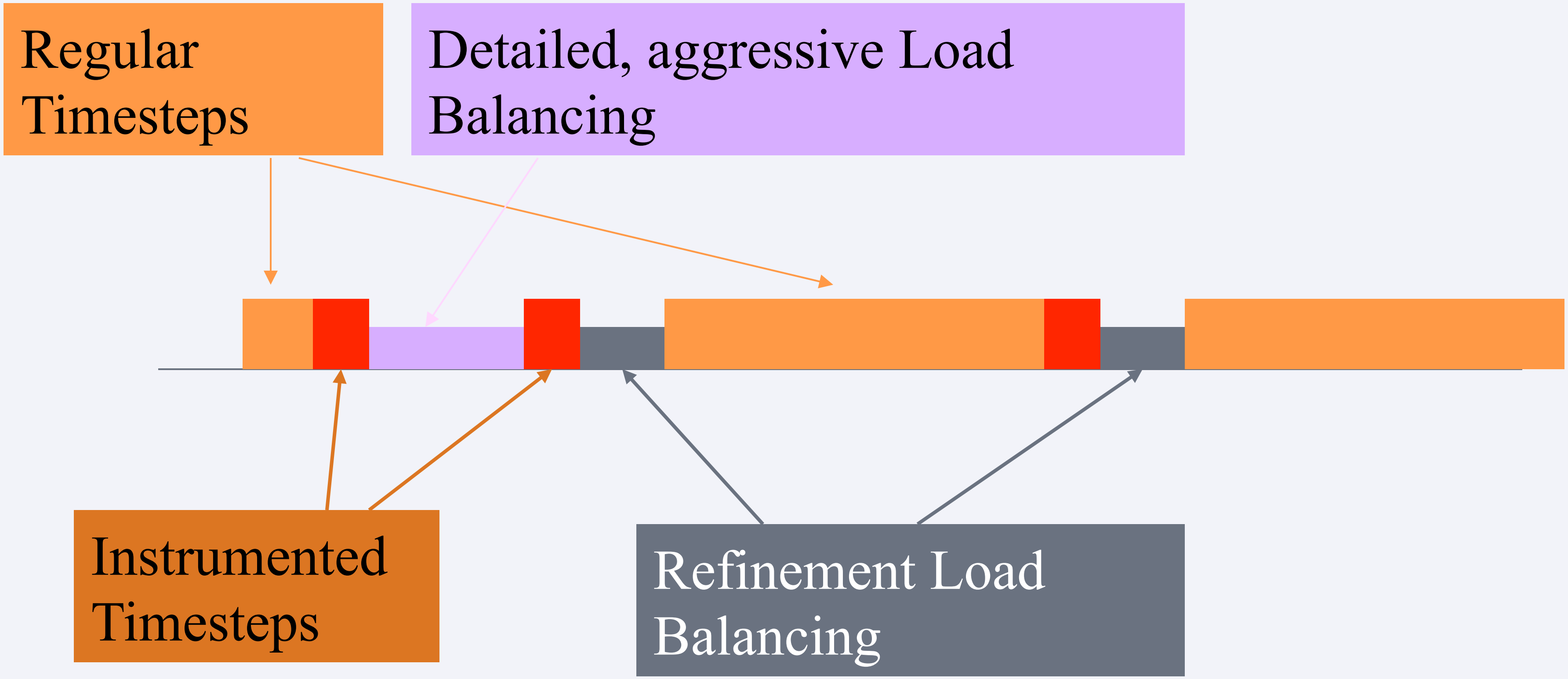
- Communication optimization
- Load balancing
- Meta-balancer
- Heterogeneous Load balancing
- Power/temperature/energy optimizations
- Resilience
- Shrink/Expand sets of nodes
- Application reconfiguration to add control points
- Adapting to memory capacity

Principle of Persistence

- Once the computation is expressed in terms of its natural (migratable) objects
- *Computational loads and communication patterns <u>tend to</u> persist, even in dynamic computations*
- So, recent past is a good predictor of near future

In spite of increase in irregularity and adaptivity, this principle still applies at exascale, and is our main friend.

Measurement-based Load Balancing



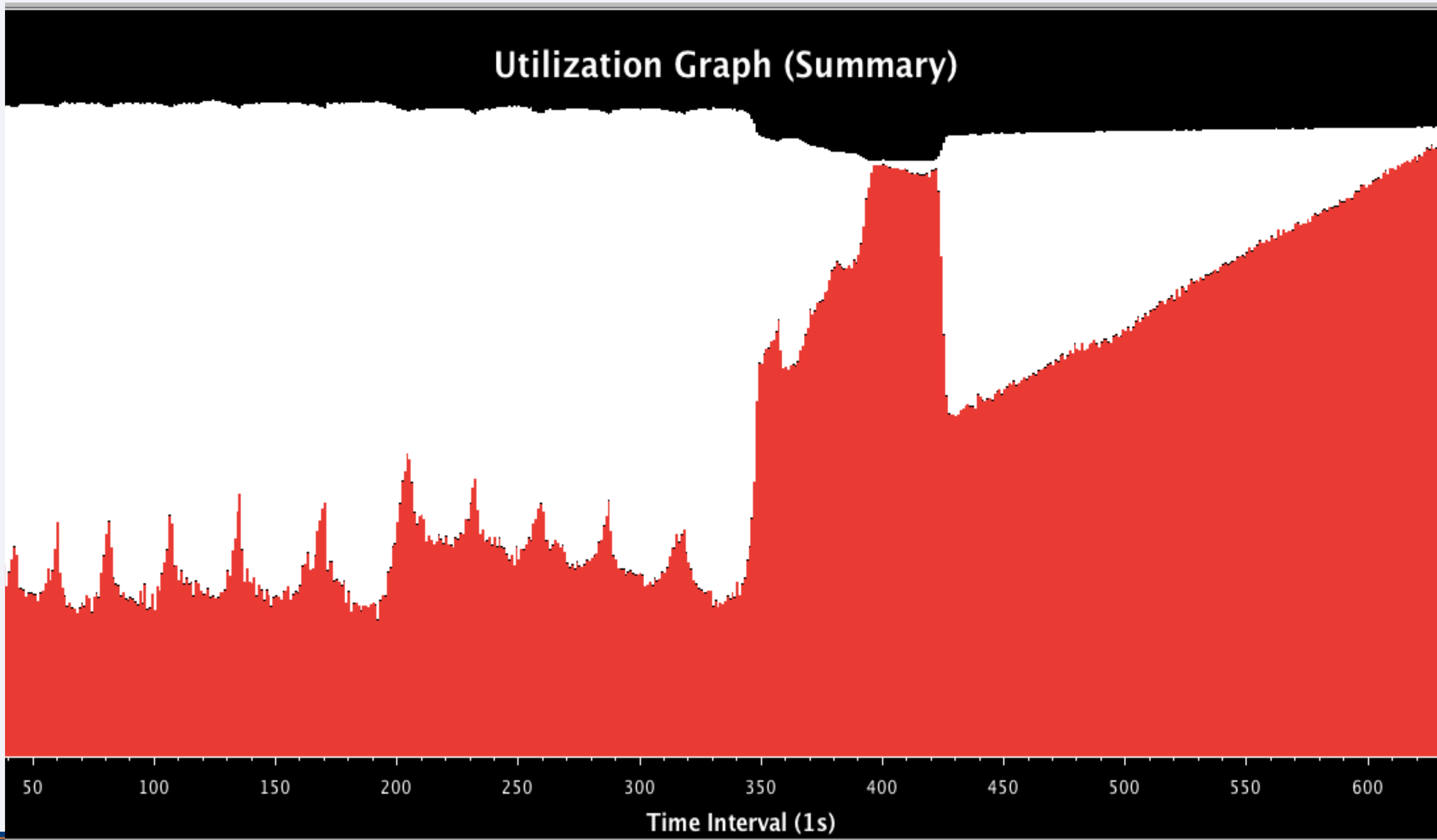
Load Balancing Framework

- Charm++ load balancing framework is an example of "customizable" RTS
- Which strategy to use, and how often to call it, can be decided for each application separately
- But if the programmer exposes one more control point, we can do more:
  - Control point: iteration boundary
  - User makes a call each iteration saying they can migrate at that point
  - Let us see what we can do: metabalancer

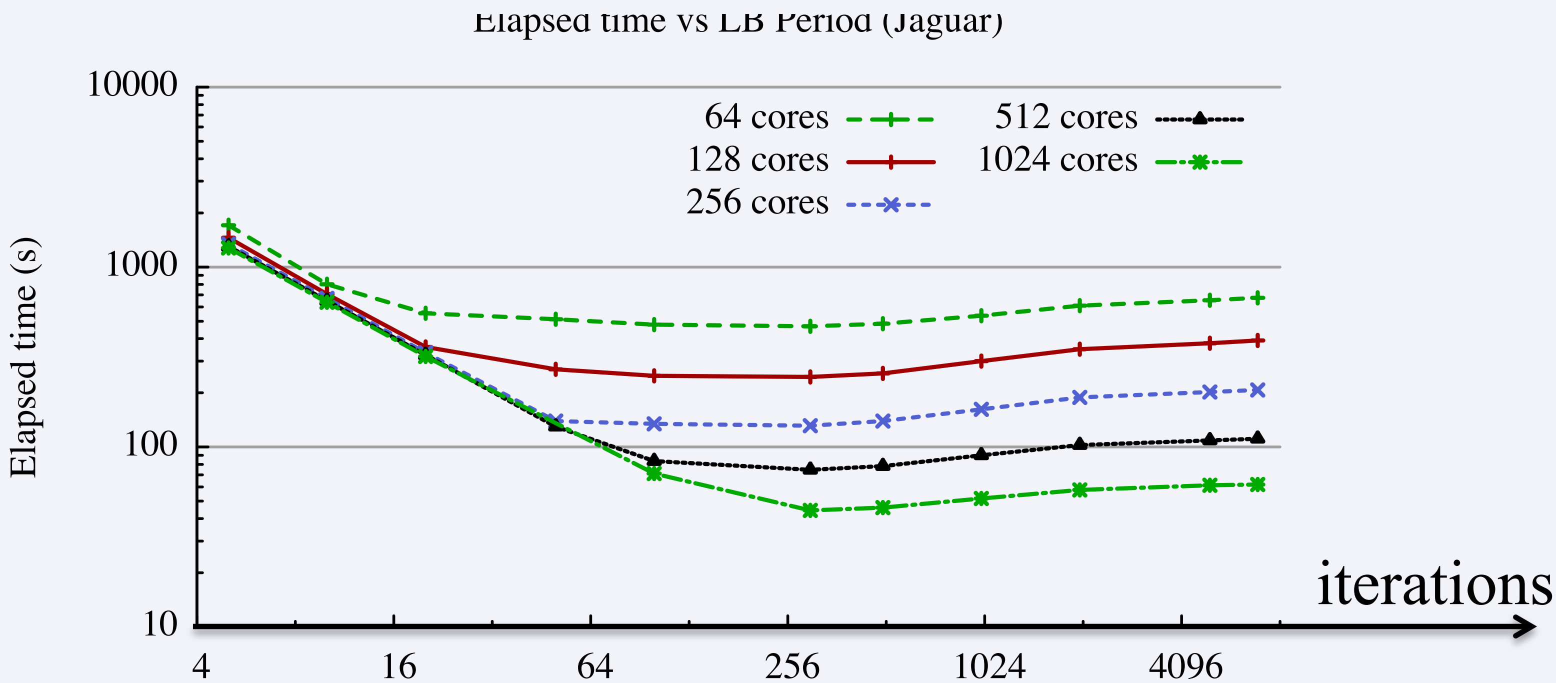
Meta-Balancer

- Automating load balancing related decision making
- Monitors the application continuously
  - Asynchronous collection of minimum statistics
- Identifies when to invoke load balancing for optimal performance based on
  - Predicted load behavior and guiding principles
  - Performance in recent past

Fractography: Without LB

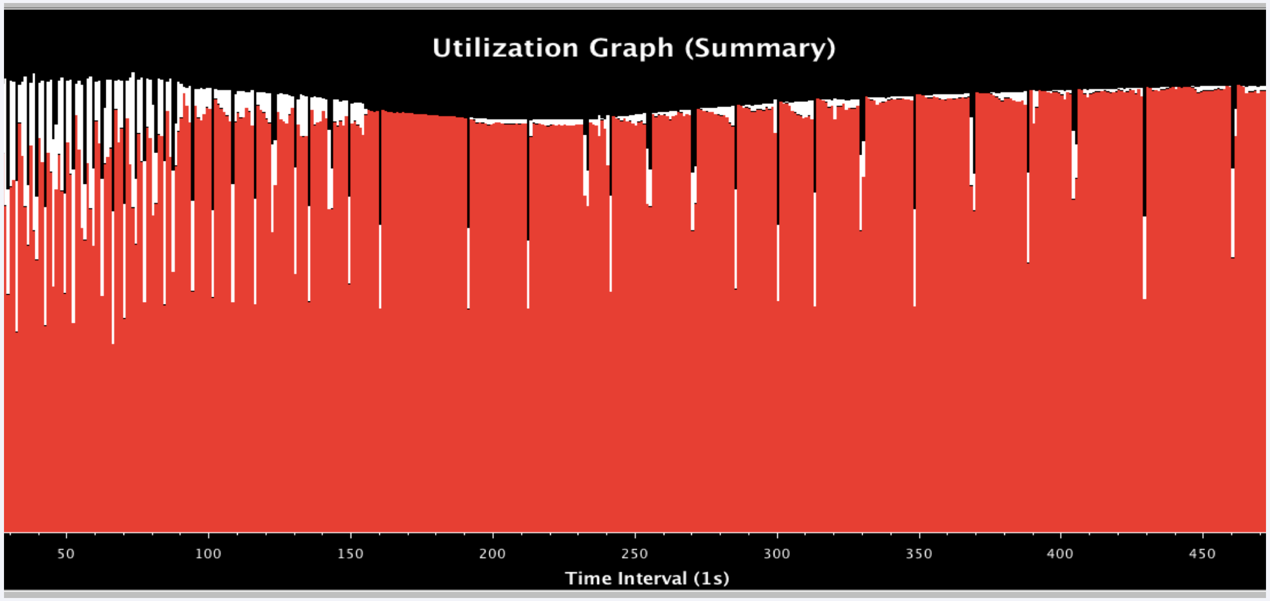


Fractography: Periodic



- Frequent load balancing leads to high overhead and no benefit
- Infrequent load balancing leads to load imbalance and results in no gains

Meta-Balancer on Fractography



- Identifies the need for frequent load balancing in the beginning
- Frequency of load balancing decreases as load becomes balanced
- Increases overall processor utilization and gives gain of 31%

Saving Cooling Energy

- Easy: increase A/C setting
  - But: some cores may get too hot
- Reduce frequency if temperature is high
  - Independently for each core or chip
- This creates a load imbalance!
- Migrate objects away from the slowed-down processors
  - Balance load using an existing strategy
  - Strategies take speed of processors into account
- Recently implemented in experimental version
  - SC 2011 paper
- Several new power/energy-related strategies

Saving Cooling Energy

- Easy: increase A/C setting
  - But: some cores may get too hot
- So, Reduce frequency if temperature is high
  - Independently for each core or chip
- *But,* This creates a load imbalance!
- No prolem, we can handle that:
  - Migrate objects away from the slowed-down Procs
  - Balance load using an existing strategy
  - Strategies take speed of processors into account
- Implemented in experimental version
  - SC 2011 paper
  - IEEE TC paper
- Several new power/energy-related strategies
  - PASA '12: Exploiting differential sensitivities of code segments to freq change

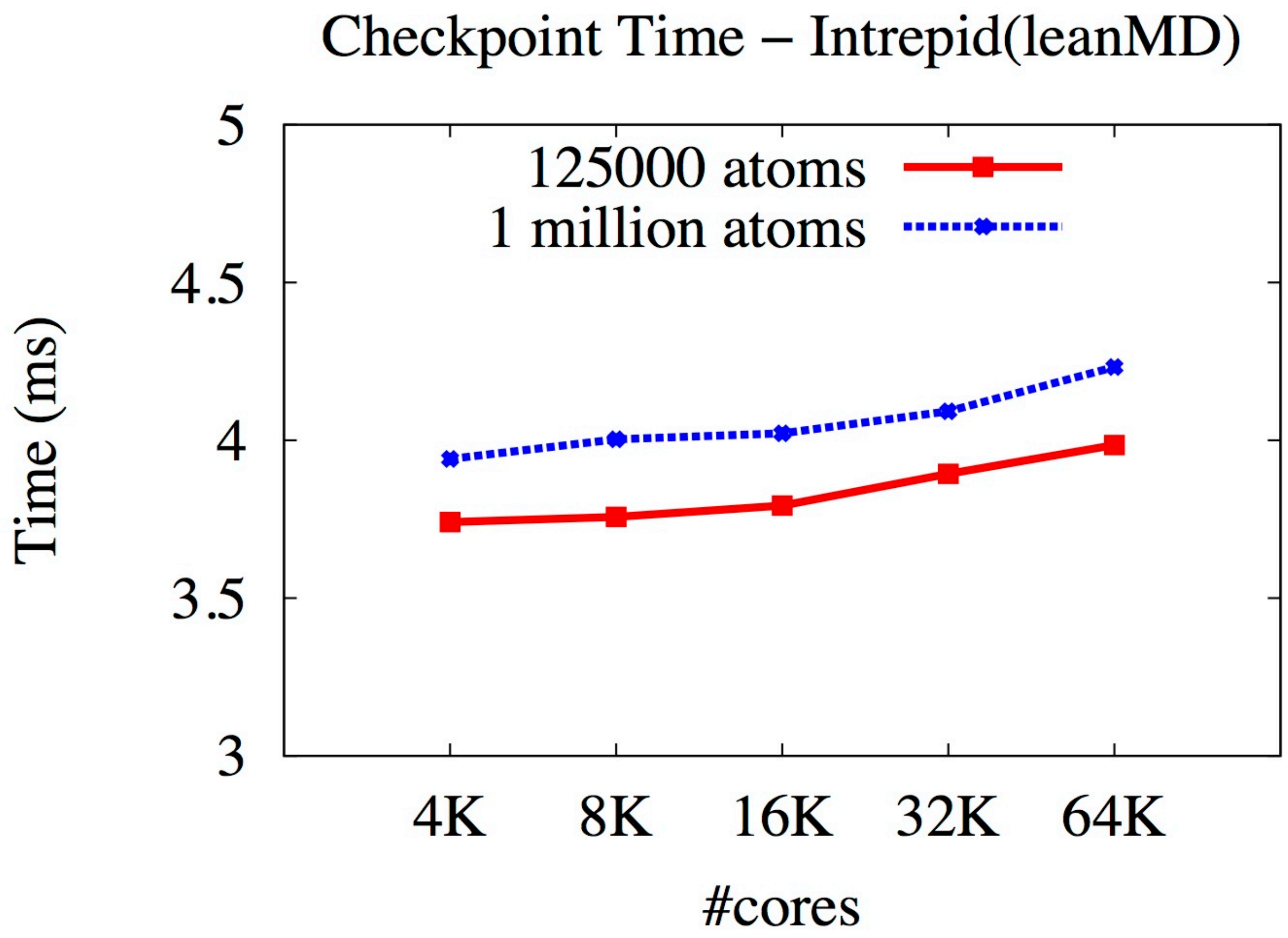
Fault Tolerance in Charm++/AMPI

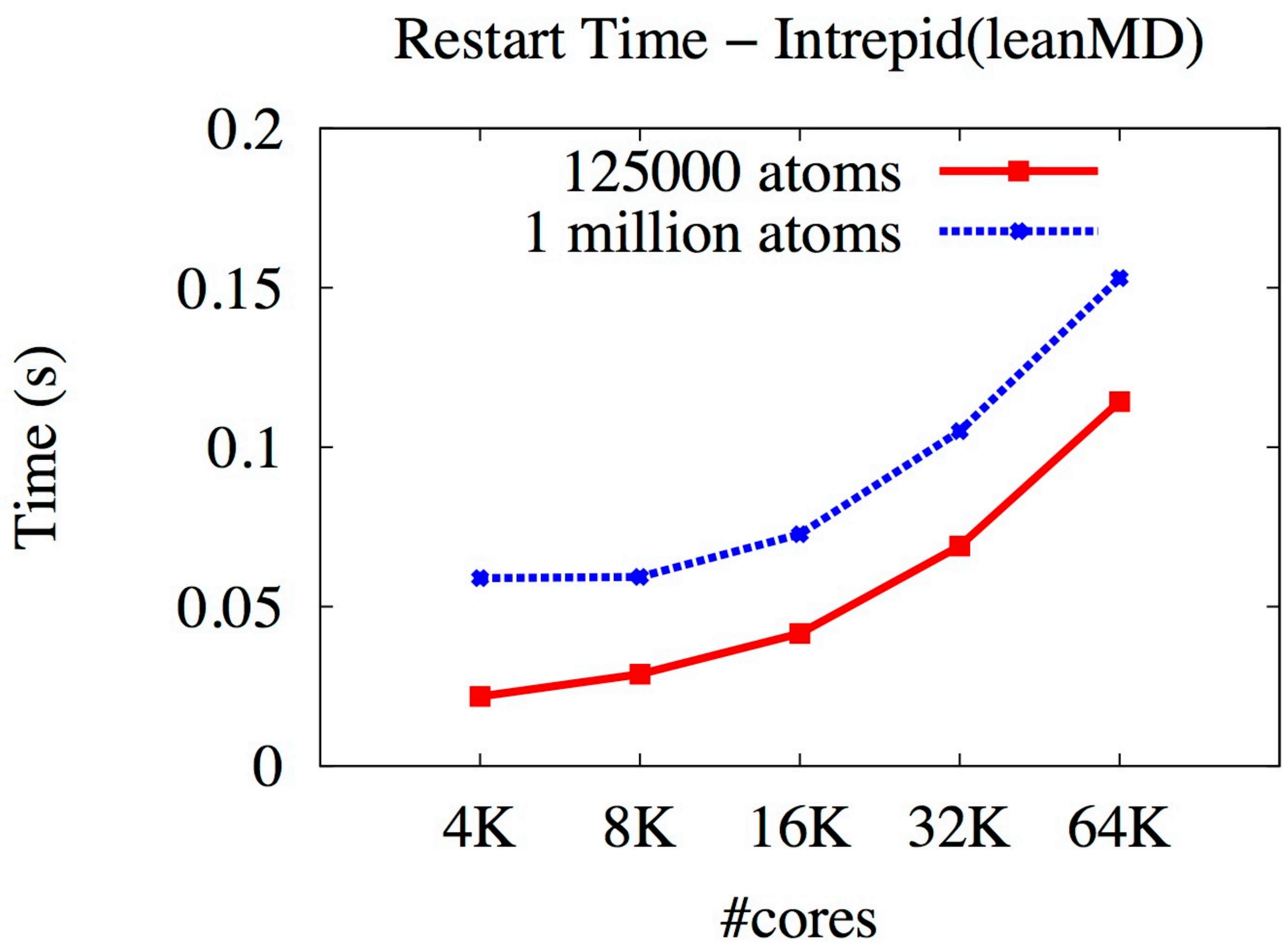
- Four Approaches:
  - Disk-based checkpoint/restart
  - In-memory double checkpoint/restart
  - Proactive object migration
  - Message-logging: scalable fault tolerance
- Common Features:
  - Leverages object-migration capabilities
  - Based on dynamic runtime capabilities
- Several new results in the last year:
  - FTXS 2012: scalability of in-mem scheme
  - Hiding checkpoint overhead .. with semi-blocking..
  - Energy efficiency of FT protocols : best paper SBAC-PAD

Ships in Charm++ distribution, for years

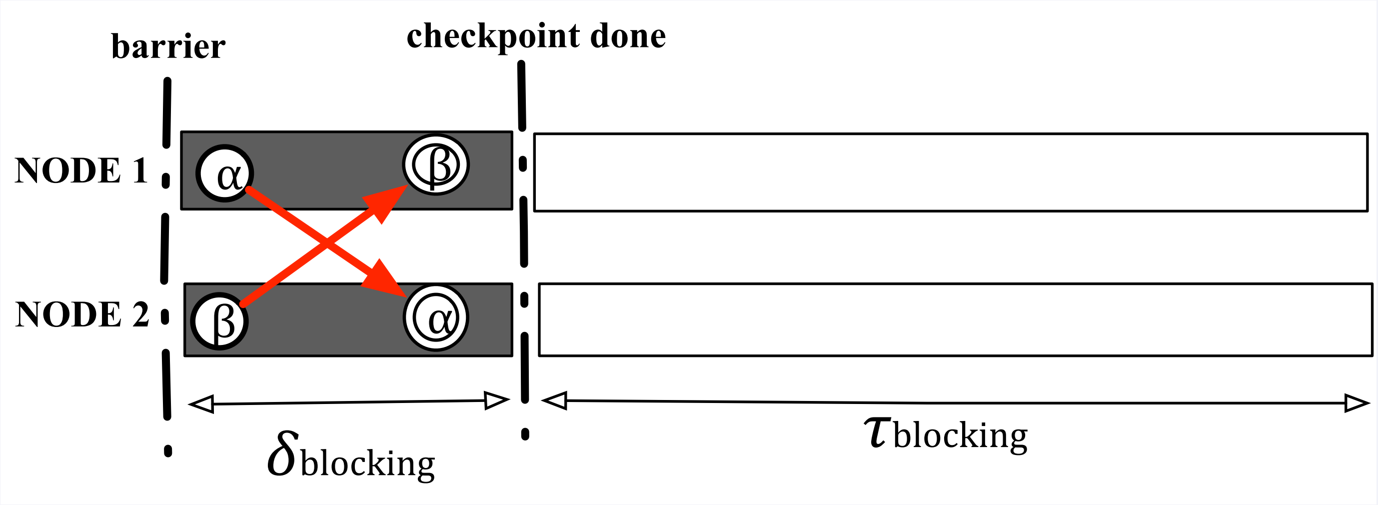
In-memory double checkpointing

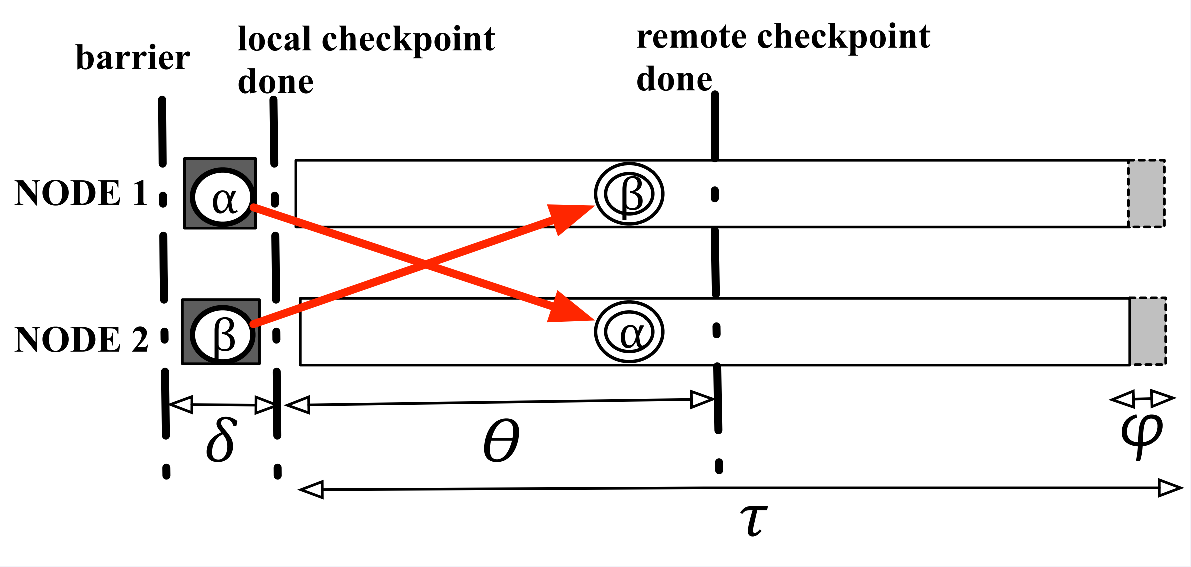
- Is practical for many apps
  - Relatively small footprint at checkpoint time
  - Also, you can use non-volatile node-local storage (e.g. FLASH)


Checkpoint Time – Intrepid(leanMD)
125000 atoms
1 million atoms
Time (ms)
5
4.5
4
3.5
3
4K
8K
16K
32K
64K
#cores

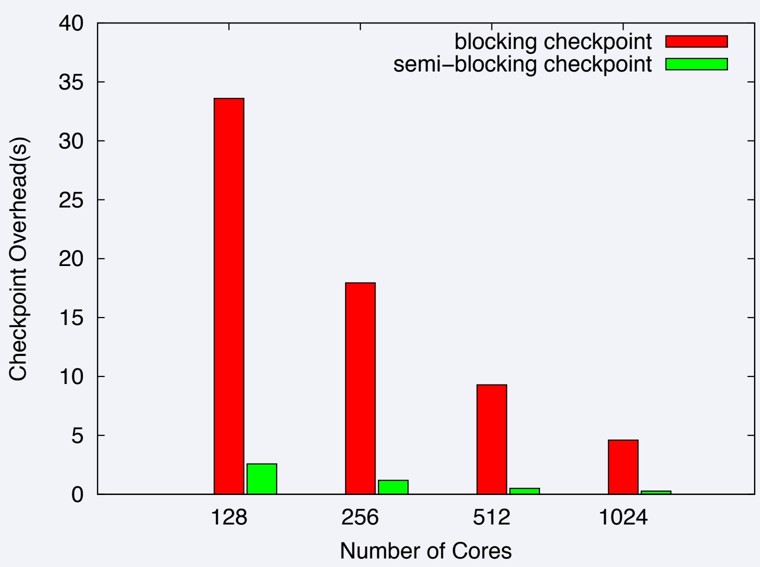

Restart Time – Intrepid(leanMD)
0.2
0.15
0.1
0.05
0
Time (s)
125000 atoms
1 million atoms
4K
8K
16K
32K
64K
#cores

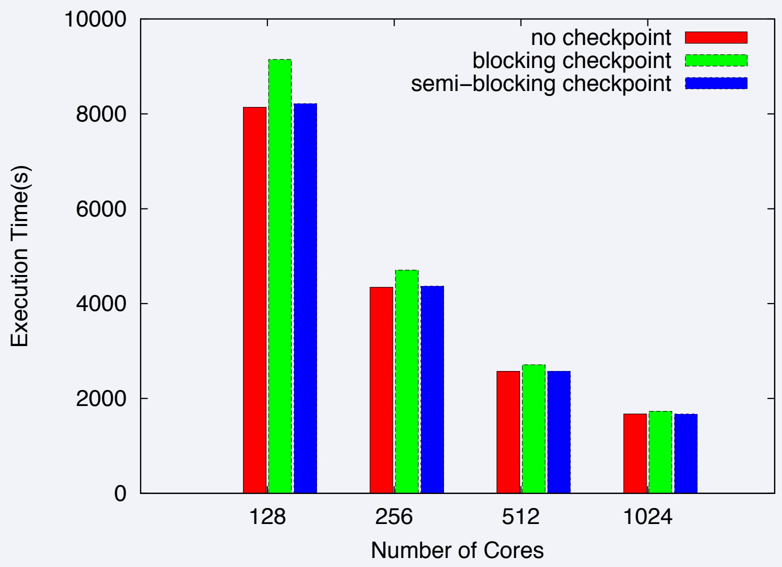
Blocking vs Semi-Blocking





Results: Strong Scaling runs of ChaNGa





The extra control exposed by the underlying communication layer was critical to attain this result

App based Creation of Control Points

- A richer set of control points can be generated if we enlist help from the application
  - Or its DSL runtime, or compiler
- The idea is:
  - Application exposes some control knobs
  - Describes the effects of the knobs
  - The RTS observes performance variables, identifies the knobs that will help the most, and turns them in the right direction
- Examples: granularity, yield frequencies in inner loops, CPU-Accelerator balance

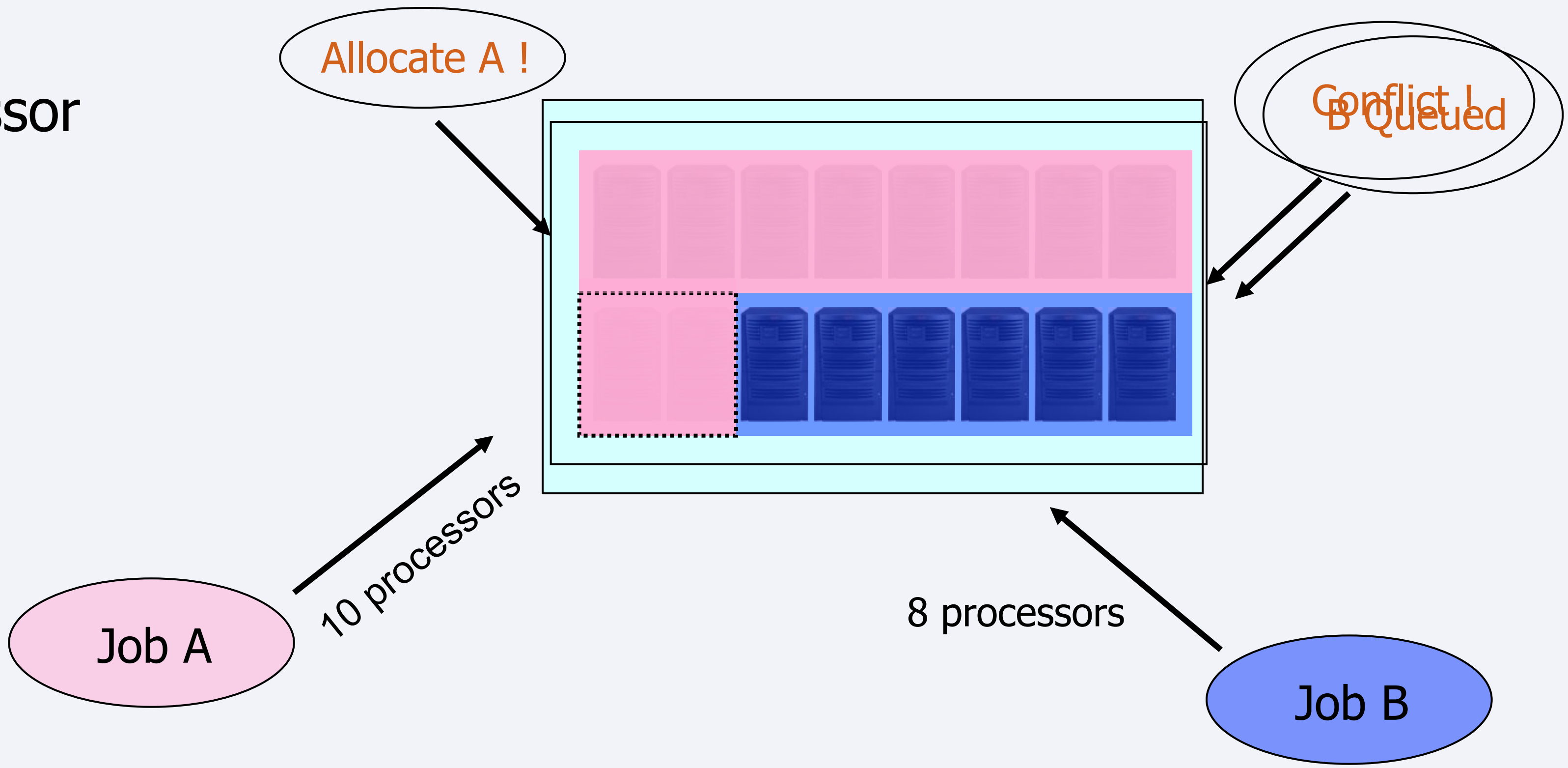
Shrink/Expand job

- If a job is told to reduce the number of nodes it is using..
- It can do so now by migrating objects..
- Same with expanding the set of nodes used
- Empowered by migratability

Inefficient Utilization within a cluster

16 Processor system

- Job A
- Job B



Current Job Schedulers can lead to low system utilization !

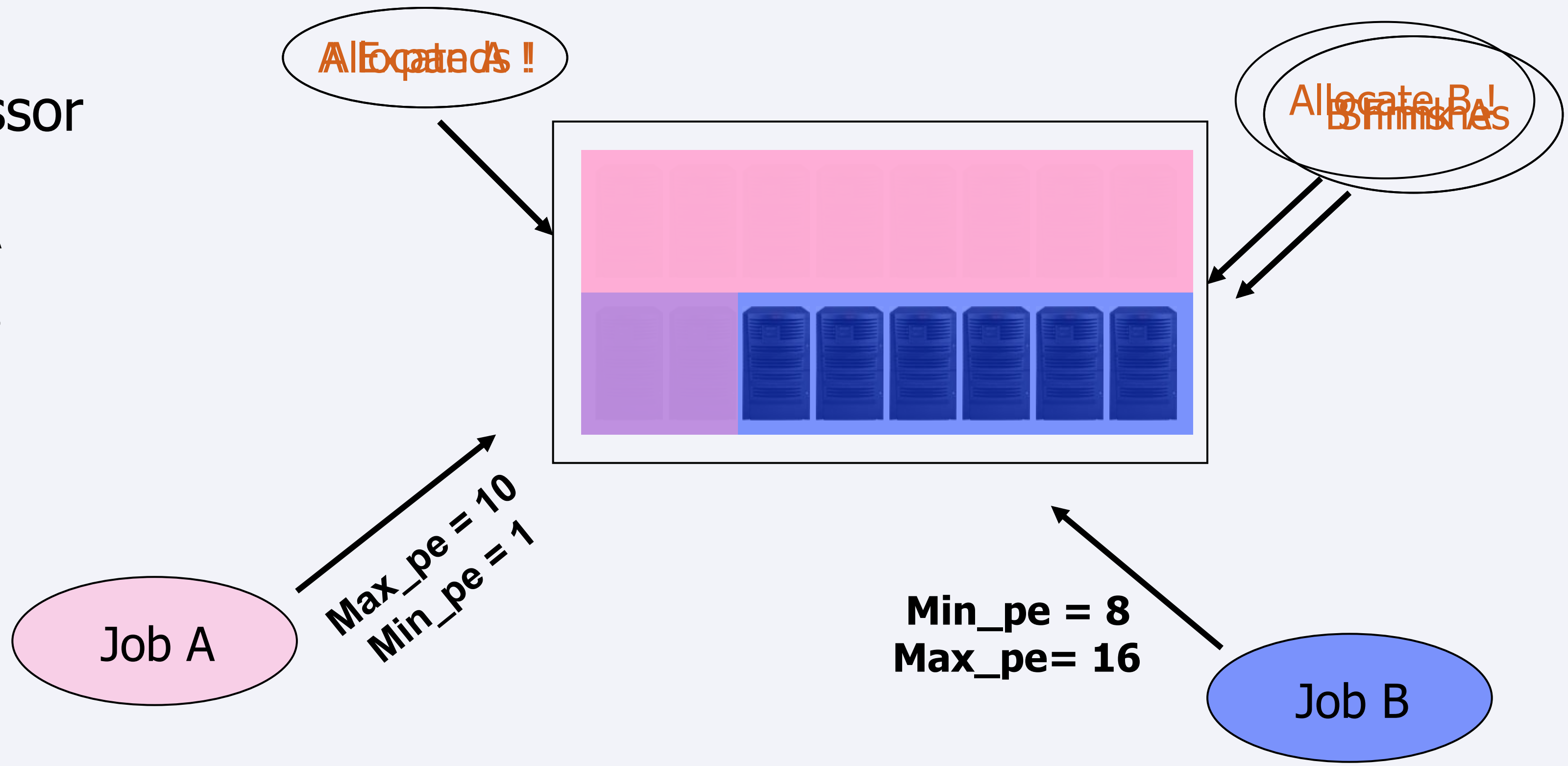
Adaptive Job Scheduler

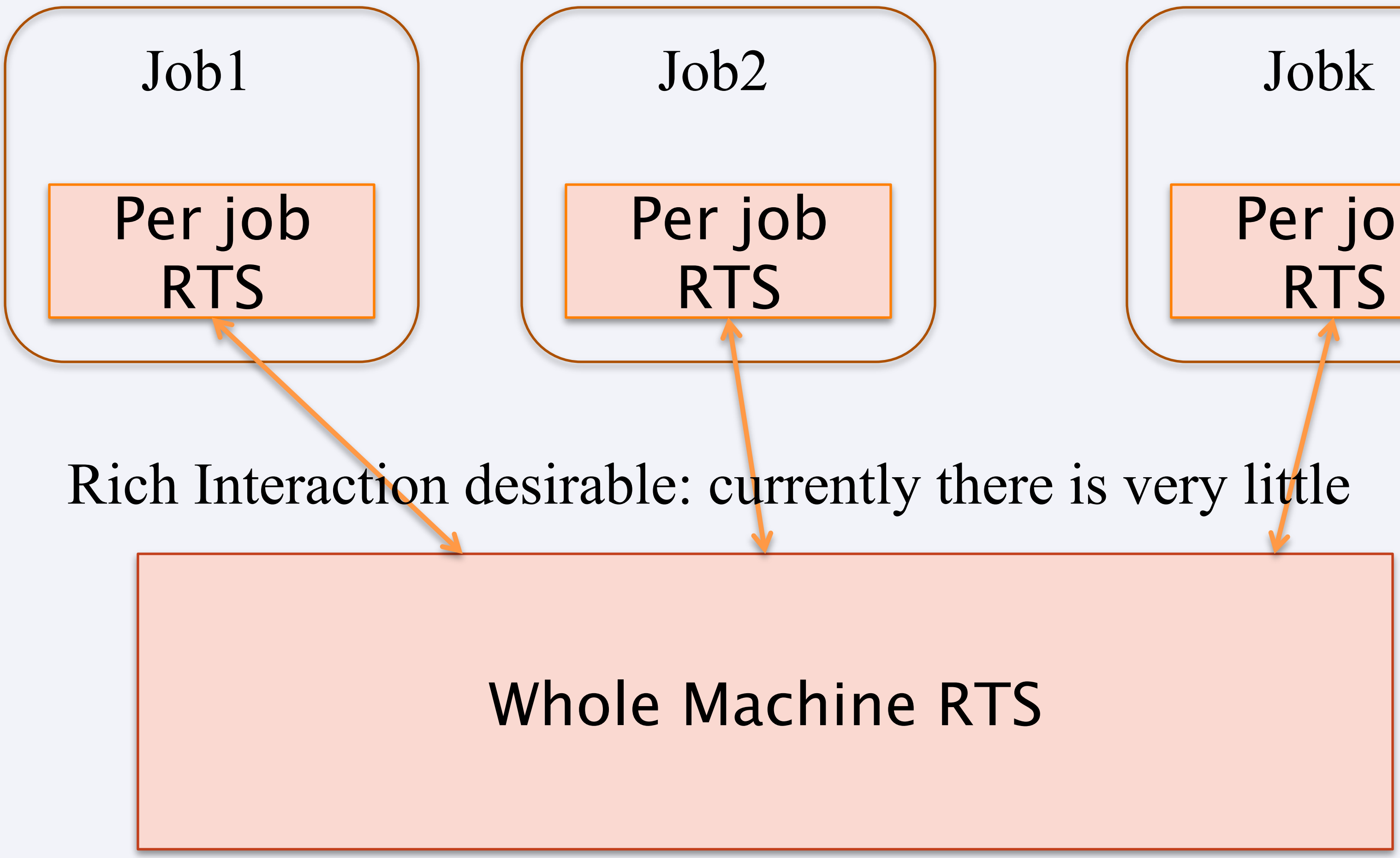
- Scheduler can take advantage of the adaptivity of AMPI and Charm++ jobs
- Improve system utilization and response time
- Scheduling decisions
  - Shrink existing jobs when a new job arrives
  - Expand jobs to use all processors when a job finishes
- Processor map sent to the job
  - Bit vector specifying which processors a job is allowed to use
    - 00011100 (use 3 4 and 5!)
- Handles regular (non-adaptive) jobs

Two Adaptive Jobs

16 Processor system

- Job A
- Job B




Job1
Per job RTS
Job2
Per job RTS
Jobk
Per job RTS
Rich Interaction desirable: currently there is very little
Whole Machine RTS

Whole machine runtime

- Job schedulers and resource allocators:
  - Accept more flexible QoS specifications from jobs
    - Creating more control variables
  - "moldable" specification:
    - This job needs between 3000-5000 nodes
    - Memory requirements..
    - Topology sensitivity, speedup profiles,…
  - Malleable:
    - this job can be told to shrink/expand after it has started

Whole machine control

- Monitor failures, and act in job-specific ways
- Global power constraints:
  - Inform, negotiate with and constrain jobs
- Thermal management
- I/O system and job I/O interactions
- Shrink and Expand jobs as needed to optimize multiple metrics

Conclusions

- We need a much richer control system
  - For each parallel job
  - For parallel machine as a whole
- Current status: paucity of control variables
- Programming models can help create new observable and controllable variables
- As far as I can see,
  - overdecomposition is the main vehicle for this..
  - Do you see other ideas?

An upcoming book
Surveys seven major applications developed using Charm++