2nd "*Scheduling in Aussois*" Workshop

Aussois, French Alps. -- May 18-21, 2008.

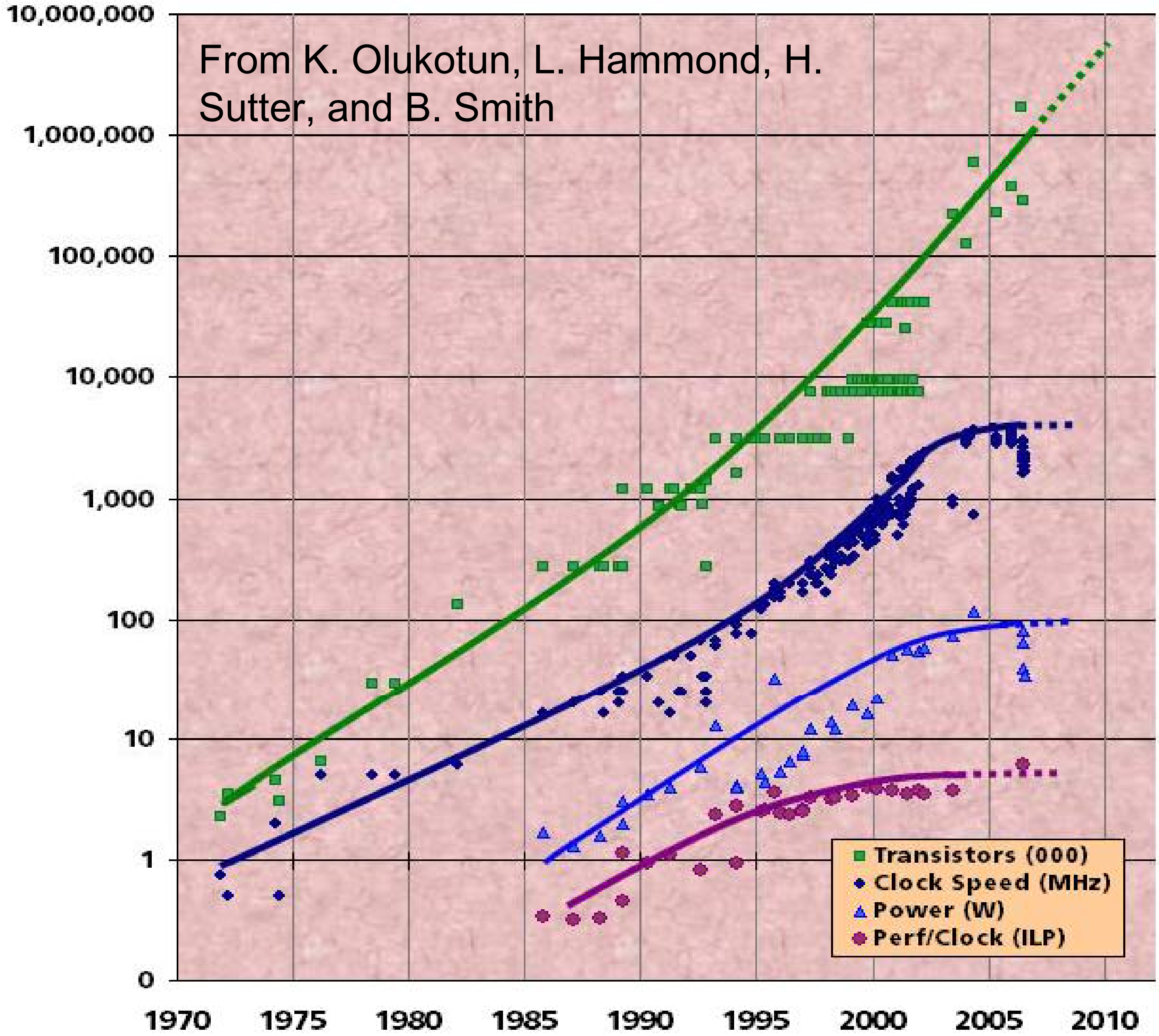
Scheduling for Numerical Linear Algebra Library at Scale

Jack Dongarra

INNOVATIVE COMPUTING LABORATORY

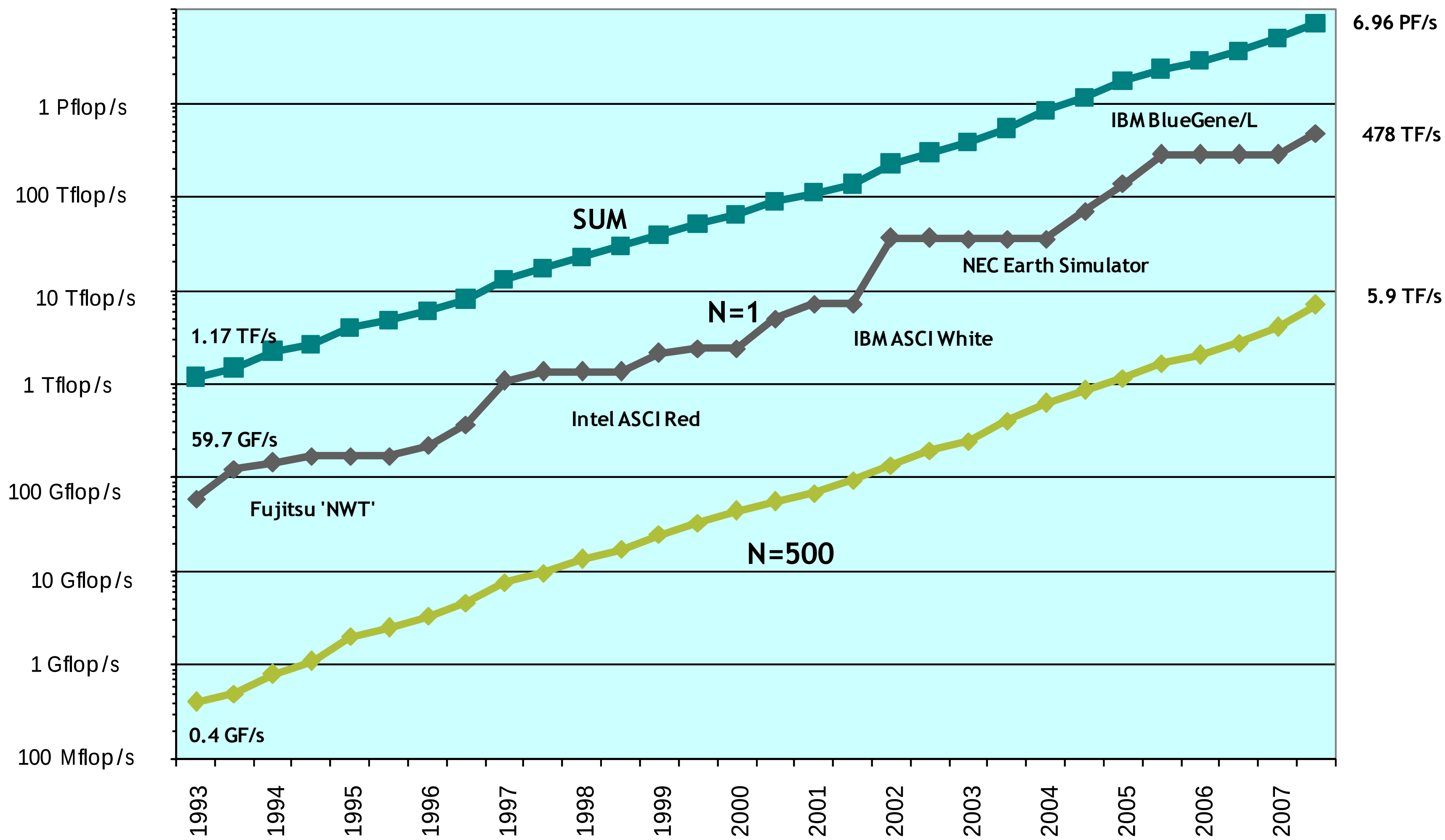
University of Tennessee
Oak Ridge National Laboratory
University of Manchester

Something's Happening Here…



- In the "old days" it was: each year processors would become faster
- Today the clock speed is fixed or getting slower
- Things are still doubling every 18 -24 months
- Moore's Law reinterpretated.
  - Number of cores double every 18-24 months

500 Fastest Computers Over Time

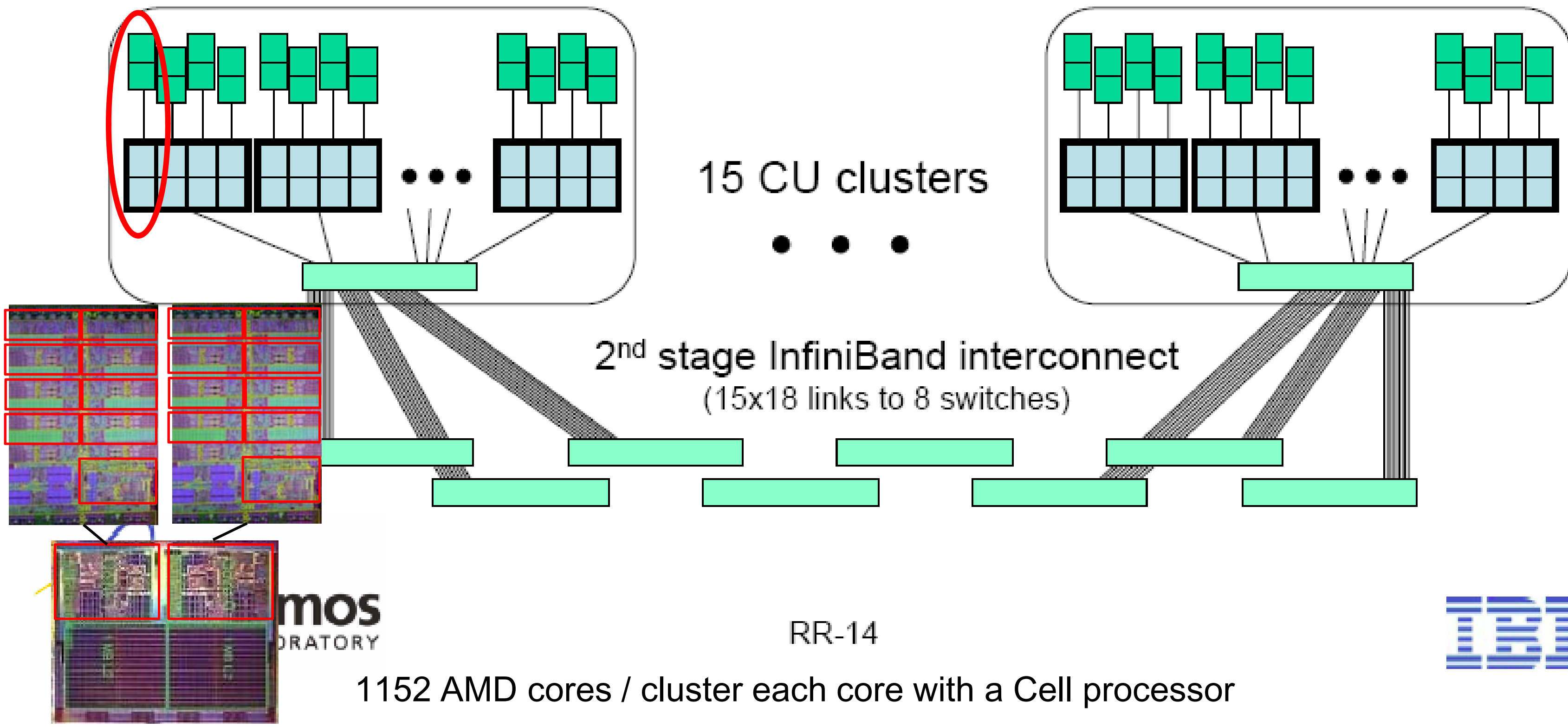


Accelerated Roadrunner

Hybrid Design (2 kinds of chips & 3 kinds of cores)

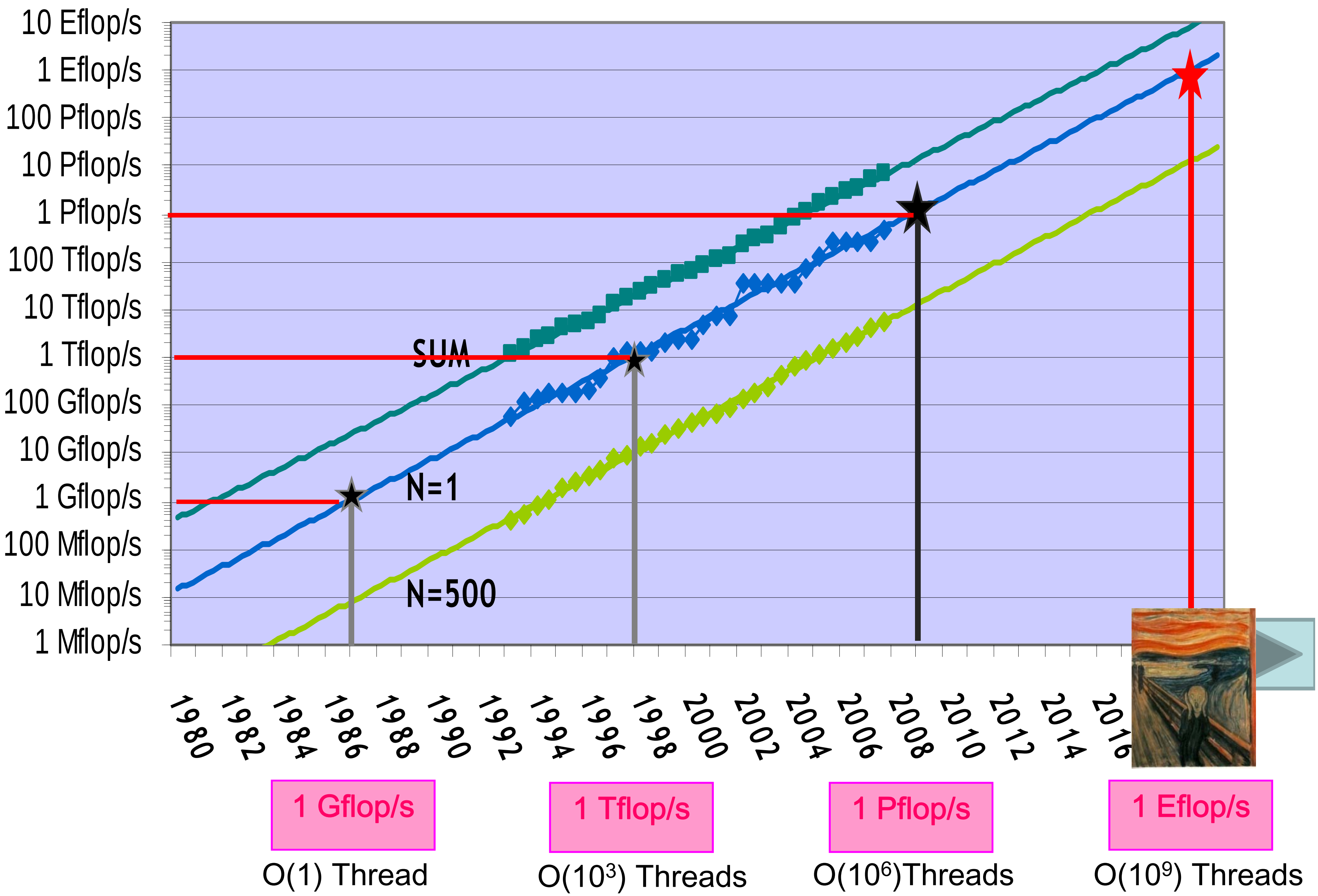
"Connected Unit" cluster
144 quad-socket
dual-core nodes
(138 w/ 4 dual-Cell blades)
InfiniBand interconnects

In aggregate:
8,640 dual-core Opterons + 16,560 eDP Cell chips
76 TeraFlops Opteron + ~1.7 PetaFlops Cell
172,800 cores



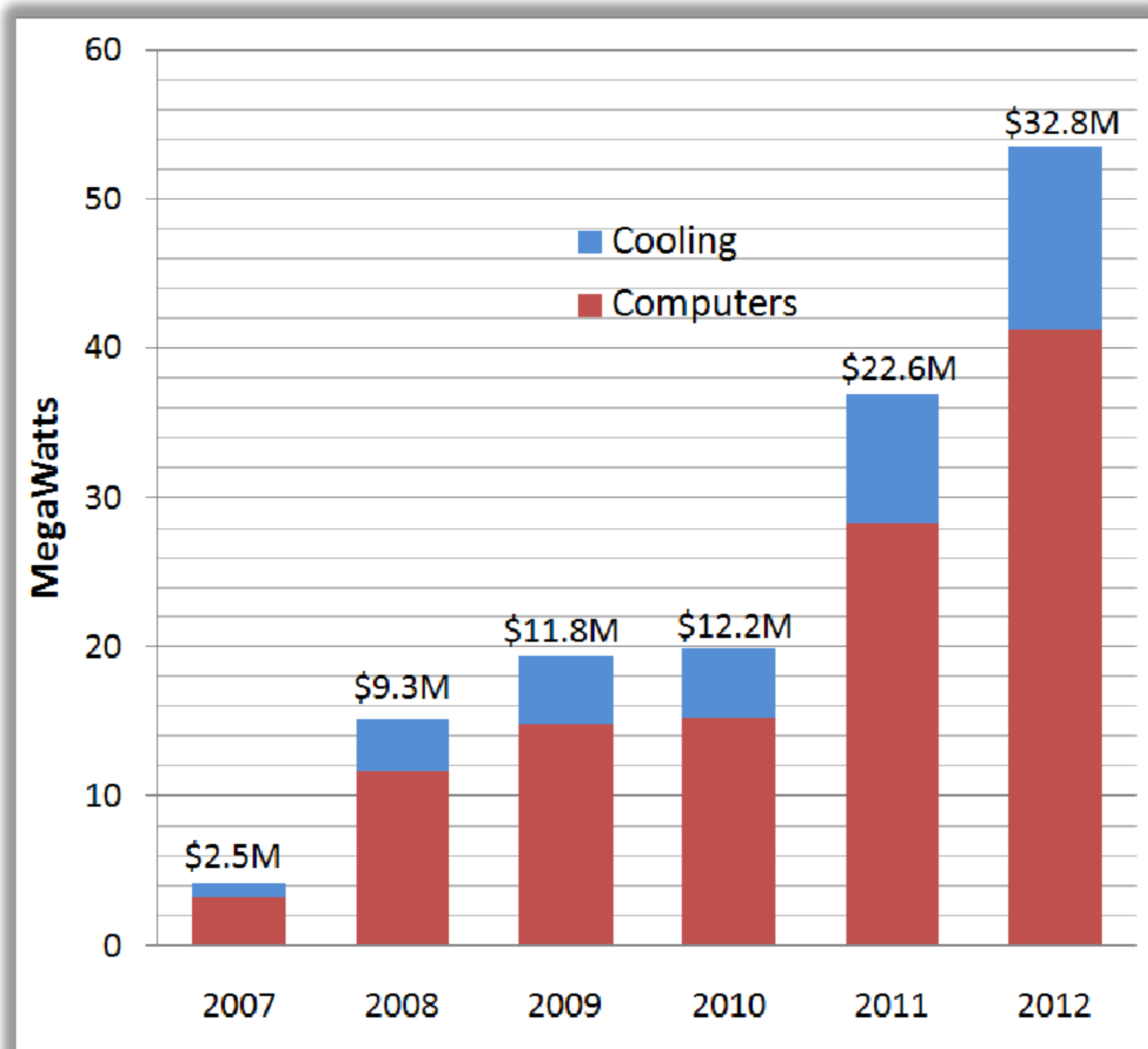
1152 AMD cores / cluster each core with a Cell processor

Performance Development & Projections



ORNL/UTK Computer Power Cost Projections 2007-2012

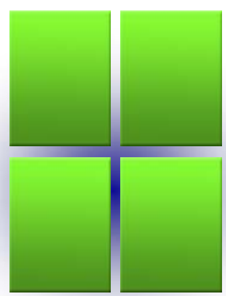
- Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems
- Using 4 MW today, going to 15MW before year end
- By 2012 could be using more than 50MW!!
- Cost estimates based on $0.07 per KwH



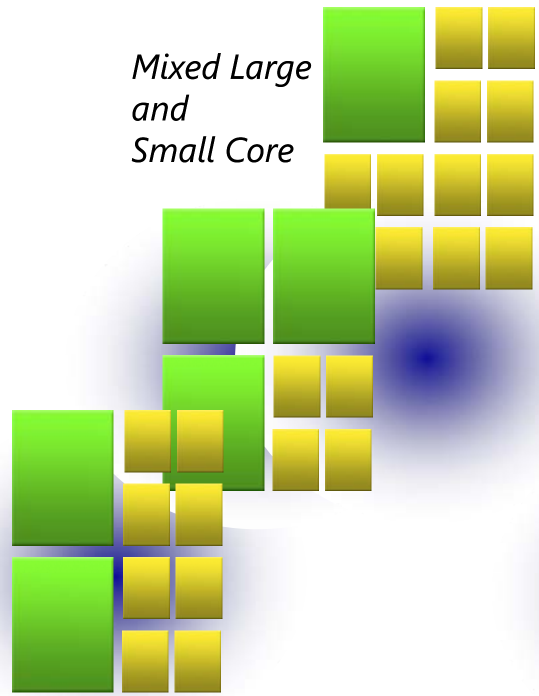
Includes both DOE and NSF systems.

What's Next?

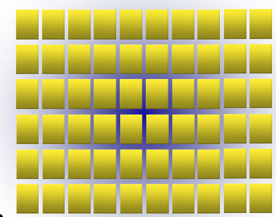
All Large Core



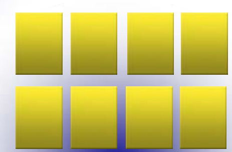
Mixed Large and Small Core



Many Small Cores

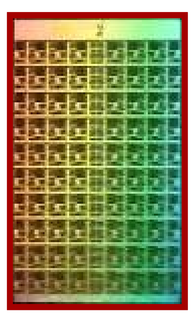


All Small Core

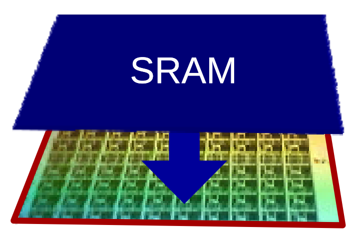


Different Classes of Chips
- Home
- Games / Graphics
- Business
- Scientific

Many Floating-Point Cores



+ 3D Stacked Memory



The question is not whether this will happen but whether we are ready

What Will a Petascale System Looks Like?

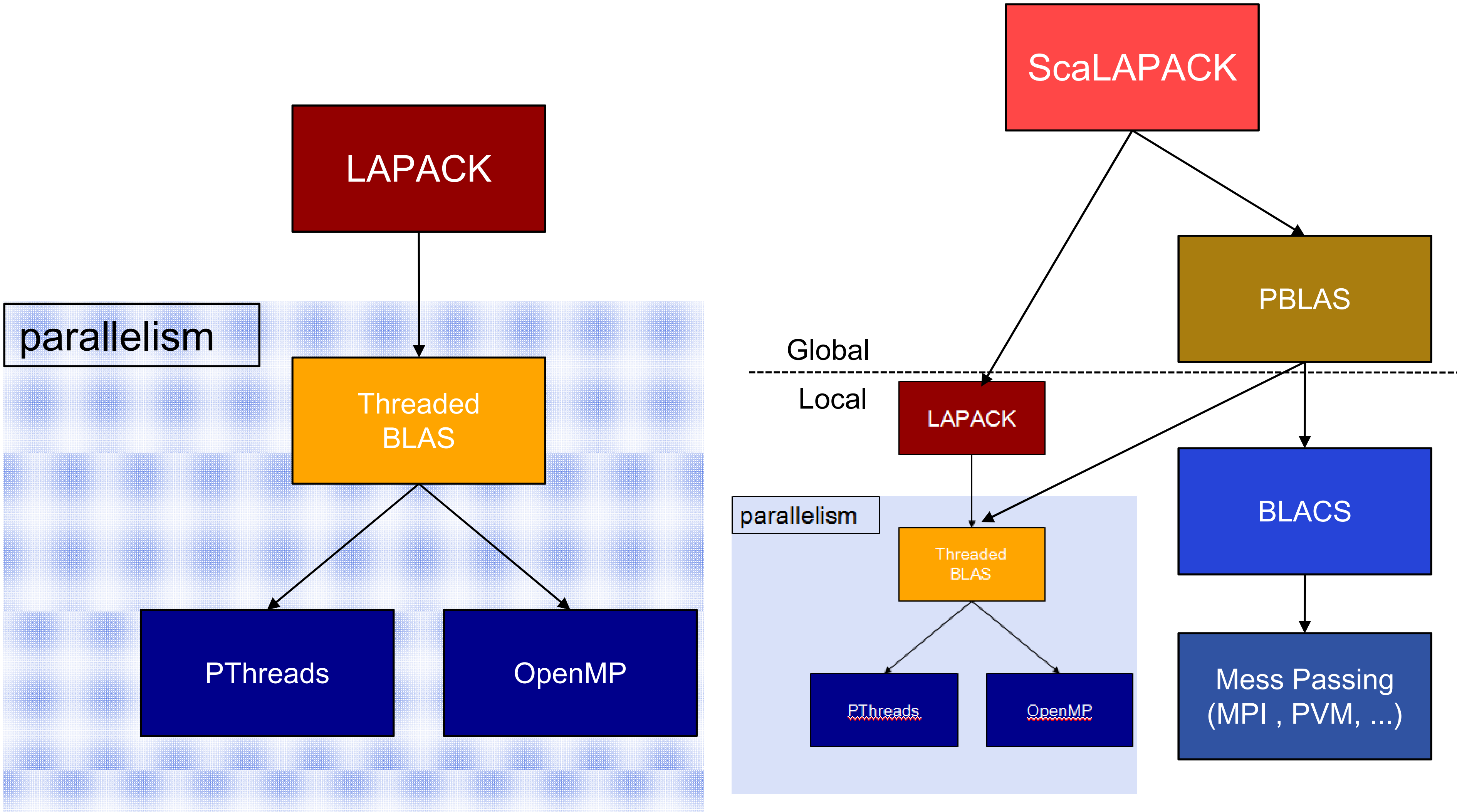
Possible Petascale System	
1. # of cores per nodes	10 - 100 cores, possibly hybrid
2. Performance per nodes	100 - 1,000 GFlop/s
3. Number of nodes	1,000 - 10,000 nodes
4. Latency inter-nodes	1 µsec
5. Bandwidth inter-nodes	10 GB/s
6. Memory per nodes	100 - 1,000 GB

- In general would like high…
  2. performance per node   5. bandwidth inter-nodes   6. memory per nodes
- Algorithms for multicore and need for latency avoiding algorithms
  1. Number of cores per node   2. performance per node   4. Latency inter-nodes
- Issues involving fault tolerance
  - Motivation in:
    1. Number of cores per node   3. number of nodes

Major Changes to Software

- **Must rethink the design of our software**
  - **Another disruptive technology**
    - **Similar to what happened with cluster computing and message passing**
  - **Rethink and rewrite the applications, algorithms, and software**
- **Numerical libraries for example will change**
  - **For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this**

LAPACK and ScaLAPACK



About 1 million lines of code

Coding for an Abstract Multicore

Parallel software for multicores should have two characteristics:

- Fine granularity:
  - High level of parallelism is needed
  - Cores will probably be associated with relatively small local memories. This requires splitting an operation into tasks that operate on small portions of data in order to reduce bus traffic and improve data locality.
- Asynchronicity:
  - As the degree of thread level parallelism grows and granularity of the operations becomes smaller, the presence of synchronization points in a parallel execution seriously affects the efficiency of an algorithm.

ManyCore - Parallelism for the Masses

- **We are looking at the following concepts in designing the next numerical library implementation**
  - **Dynamic Data Driven Execution**
  - **Self Adapting**
  - **Block Data Layout**
  - **Mixed Precision in the Algorithm**
  - **Exploit Hybrid Architectures**
  - **Fault Tolerant Methods**

Steps in the LAPACK LU

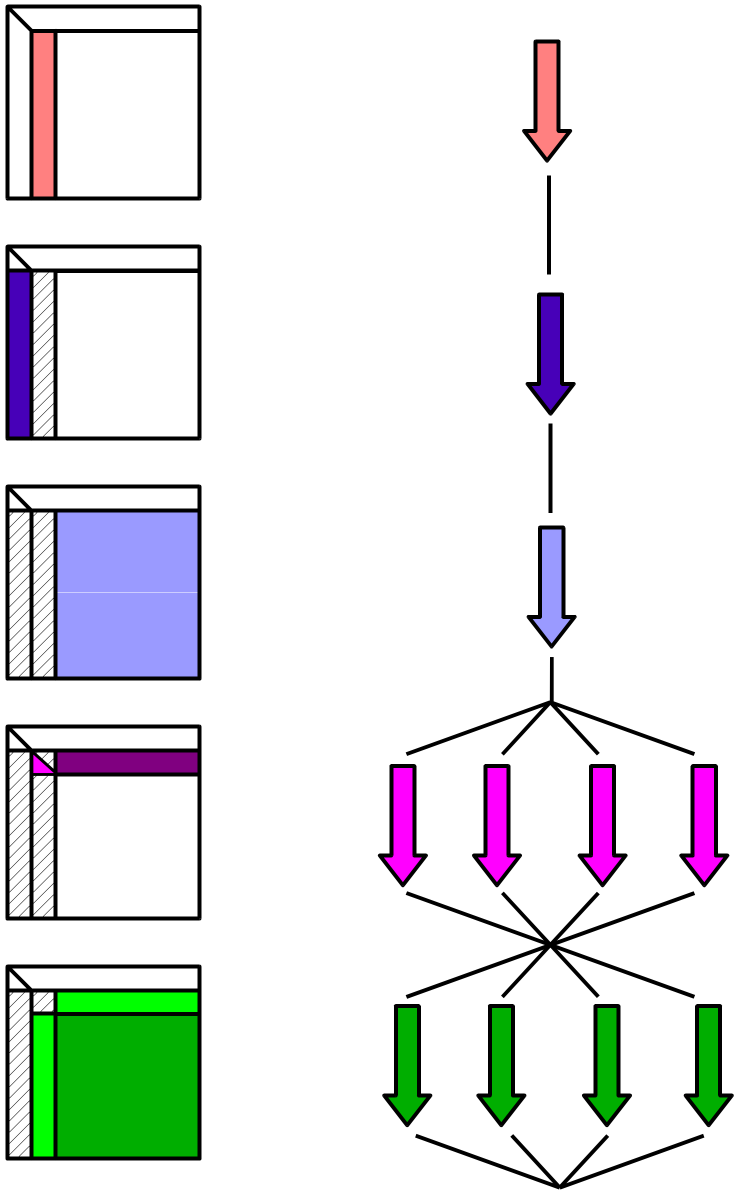
DGETF2
(Factor a panel) — LAPACK

DLSWP
(Backward swap) — LAPACK

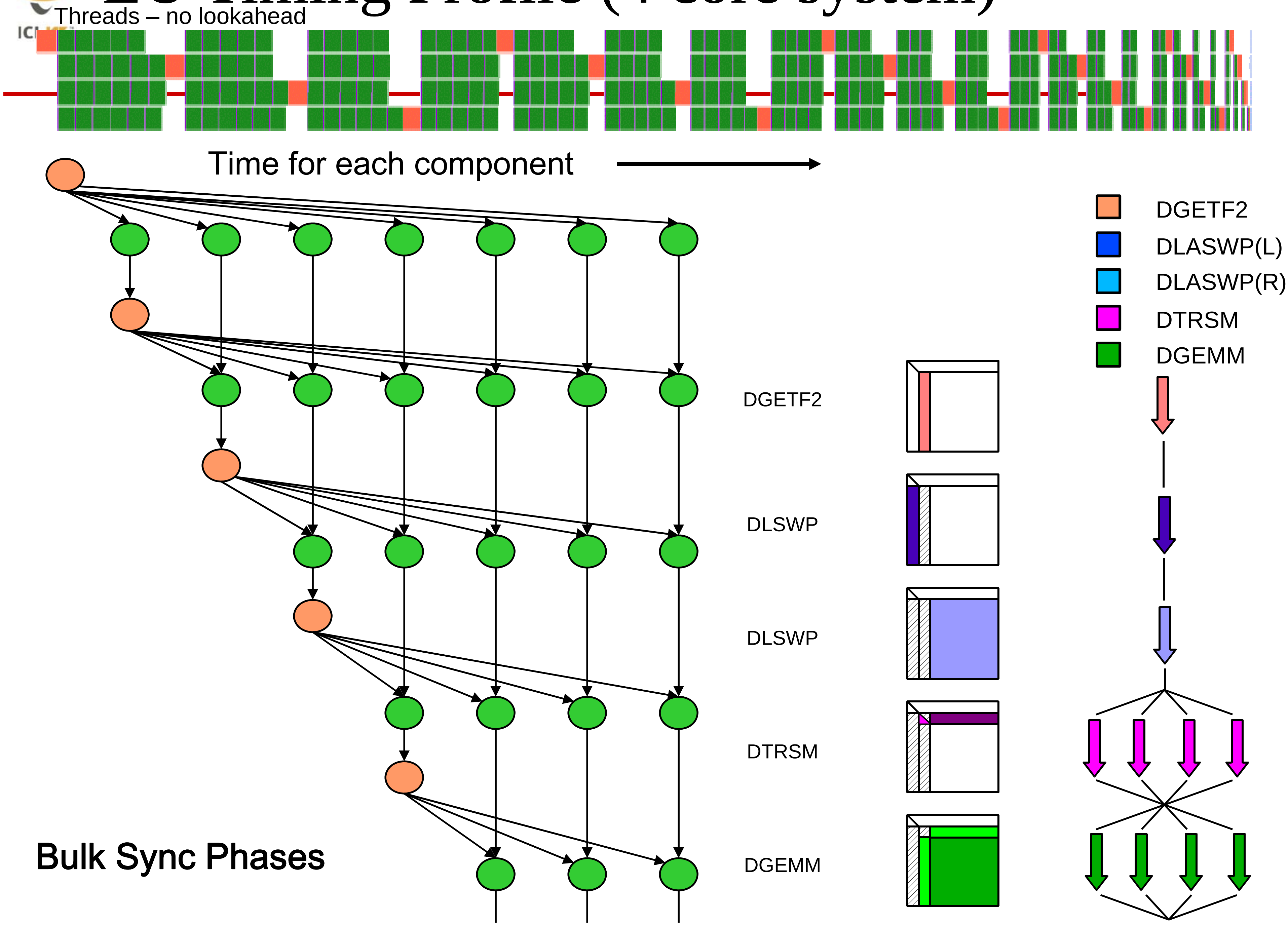
DLSWP
(Forward swap) — LAPACK

DTRSM
(Triangular solve) — BLAS

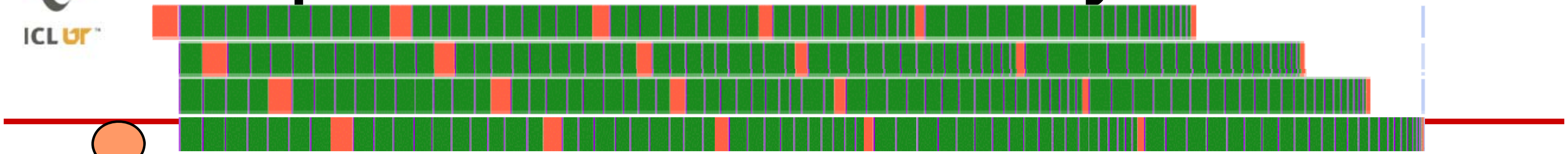
DGEMM
(Matrix multiply) — BLAS

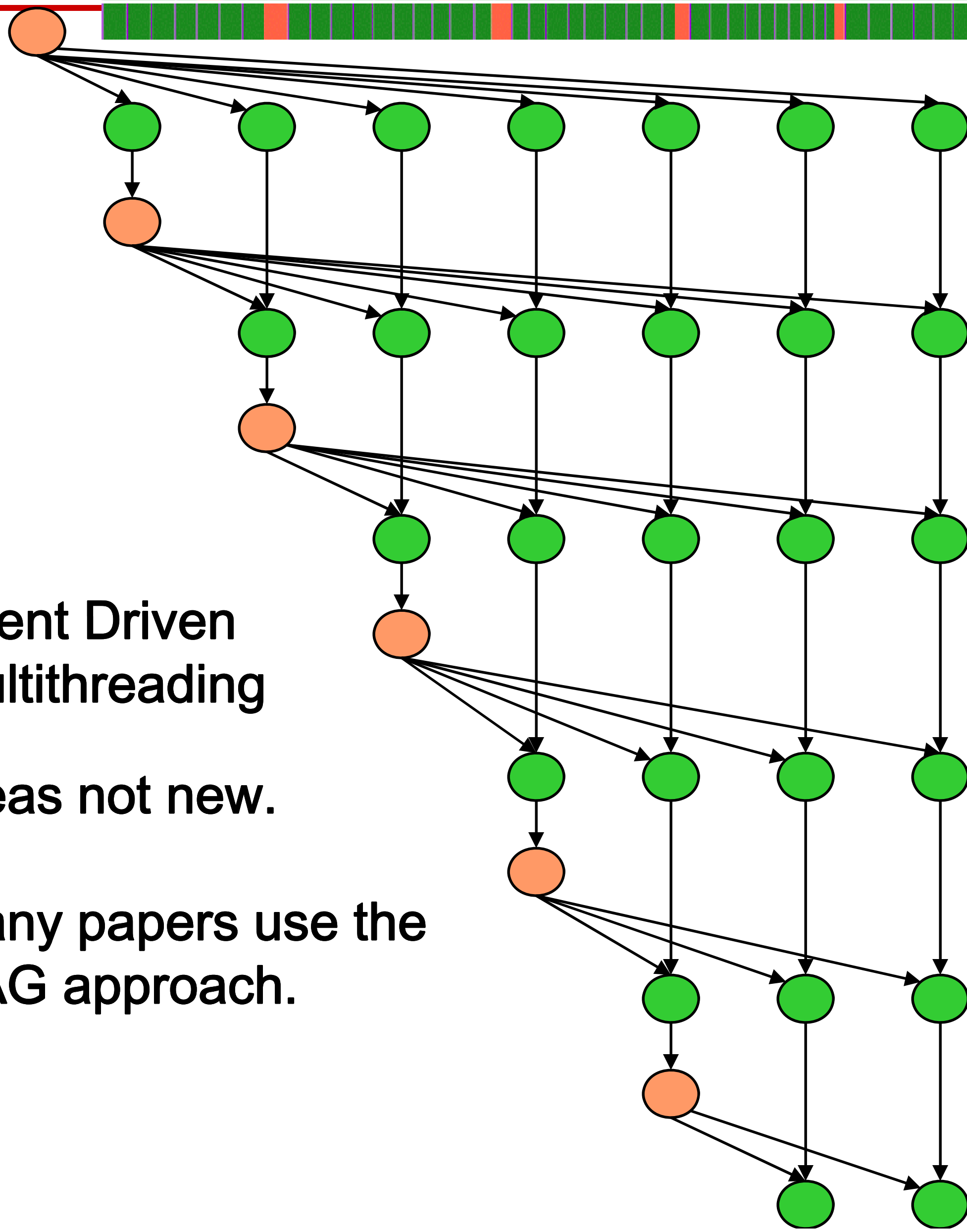


LU Timing Profile (4 core system)



Adaptive Lookahead - Dynamic





Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

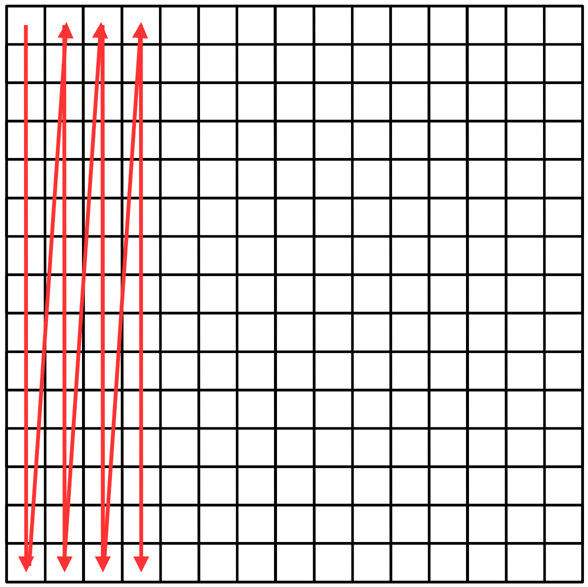
```
while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress();
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress();
        case END:
            for()
                dlaswp();
            return;
    }
}
```

Reorganizing algorithms to use this approach

Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

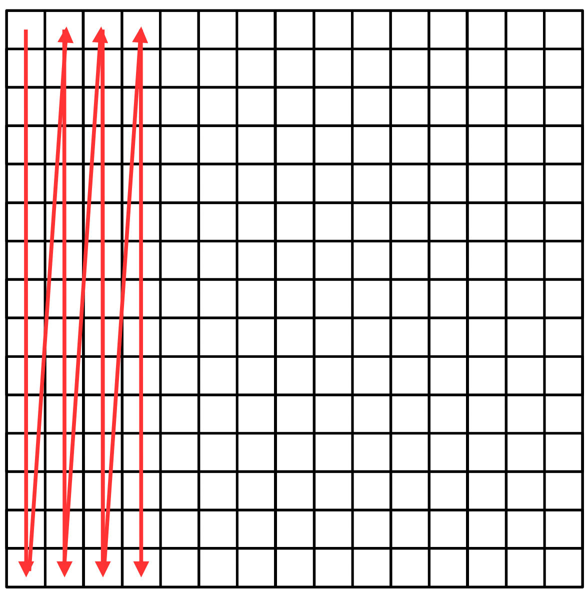
Column-Major



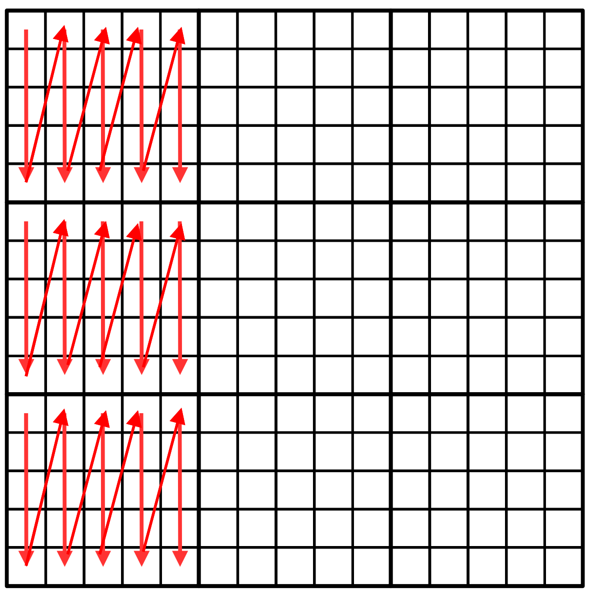
Achieving Fine Granularity

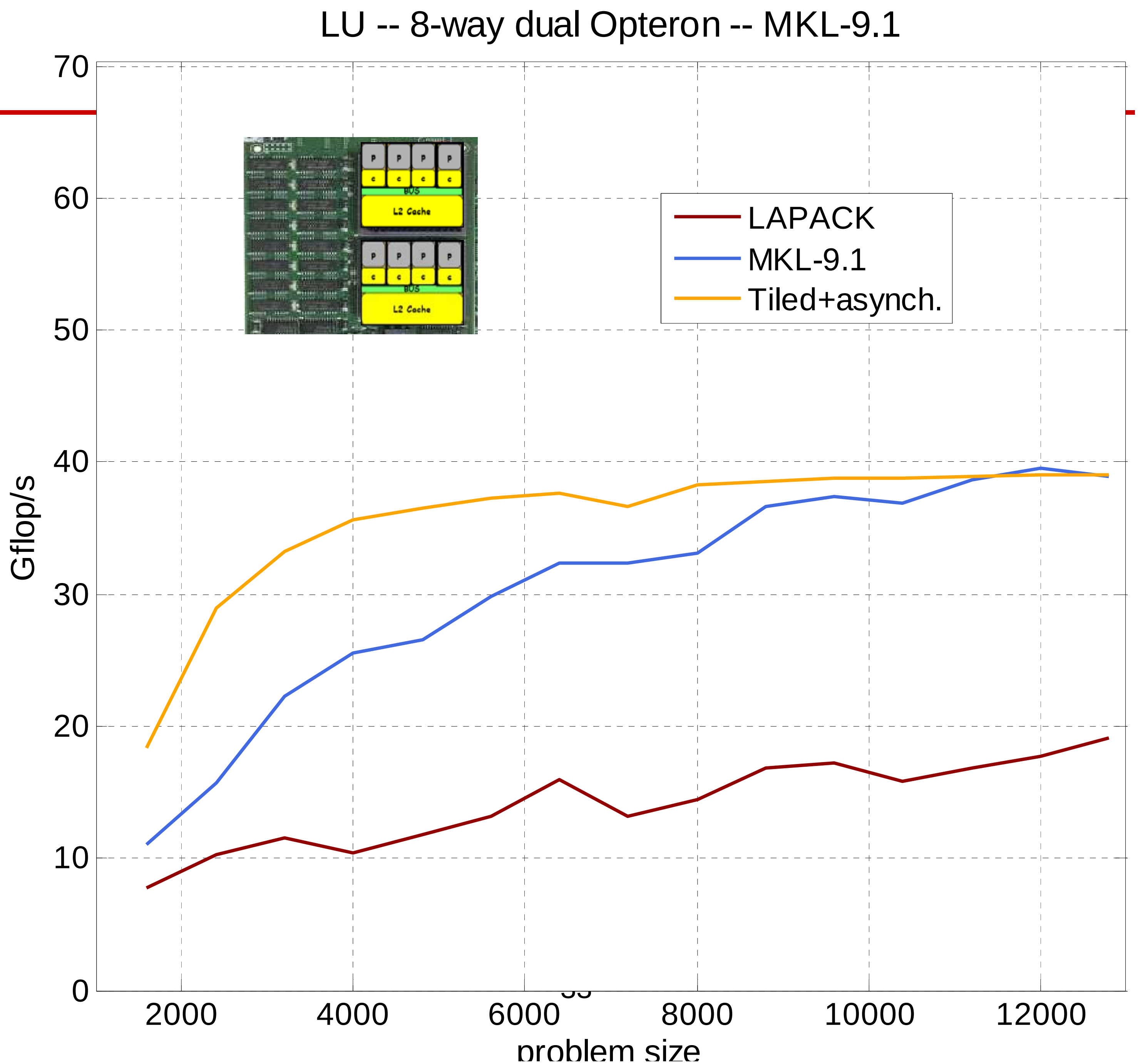
Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

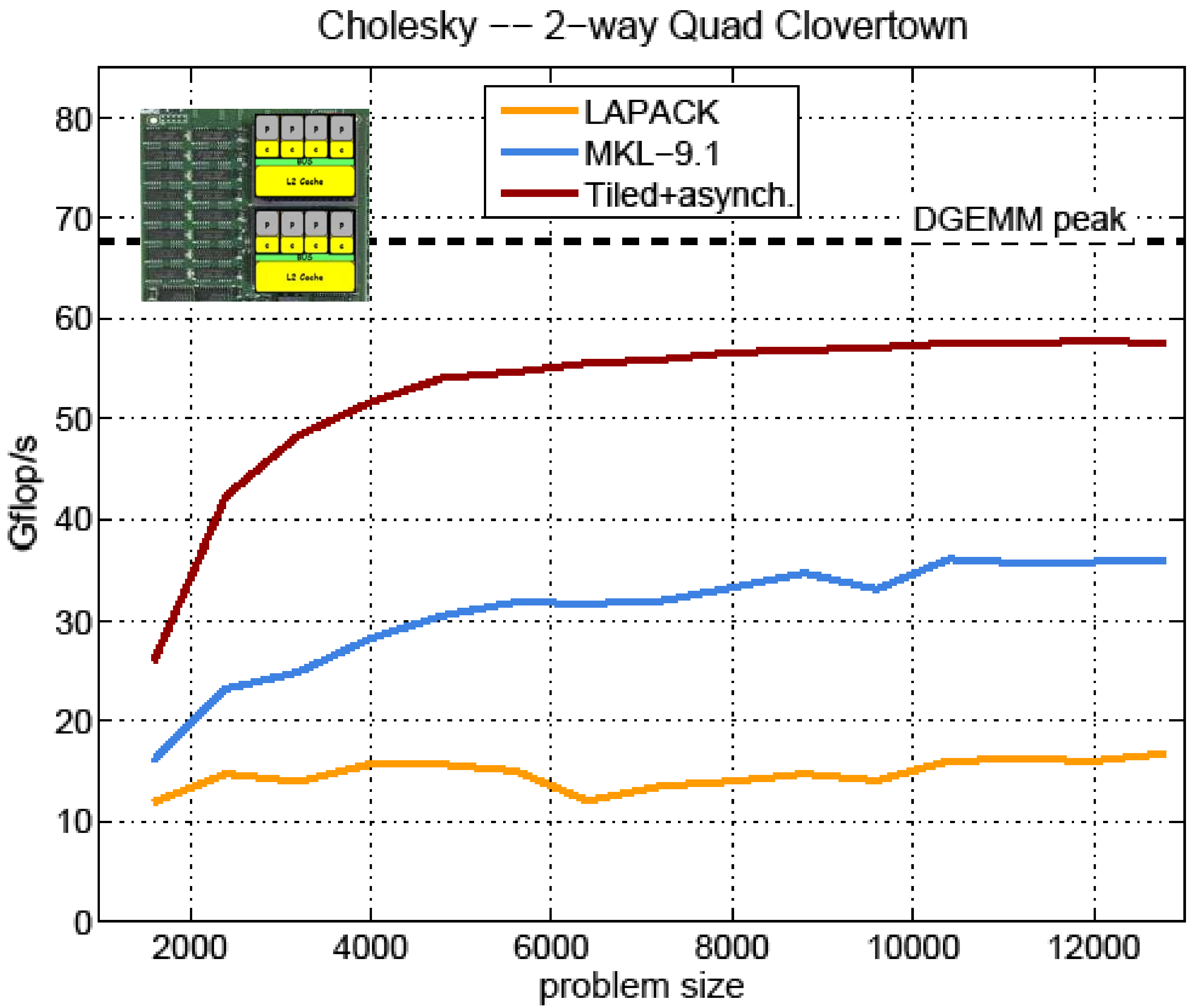
Column-Major



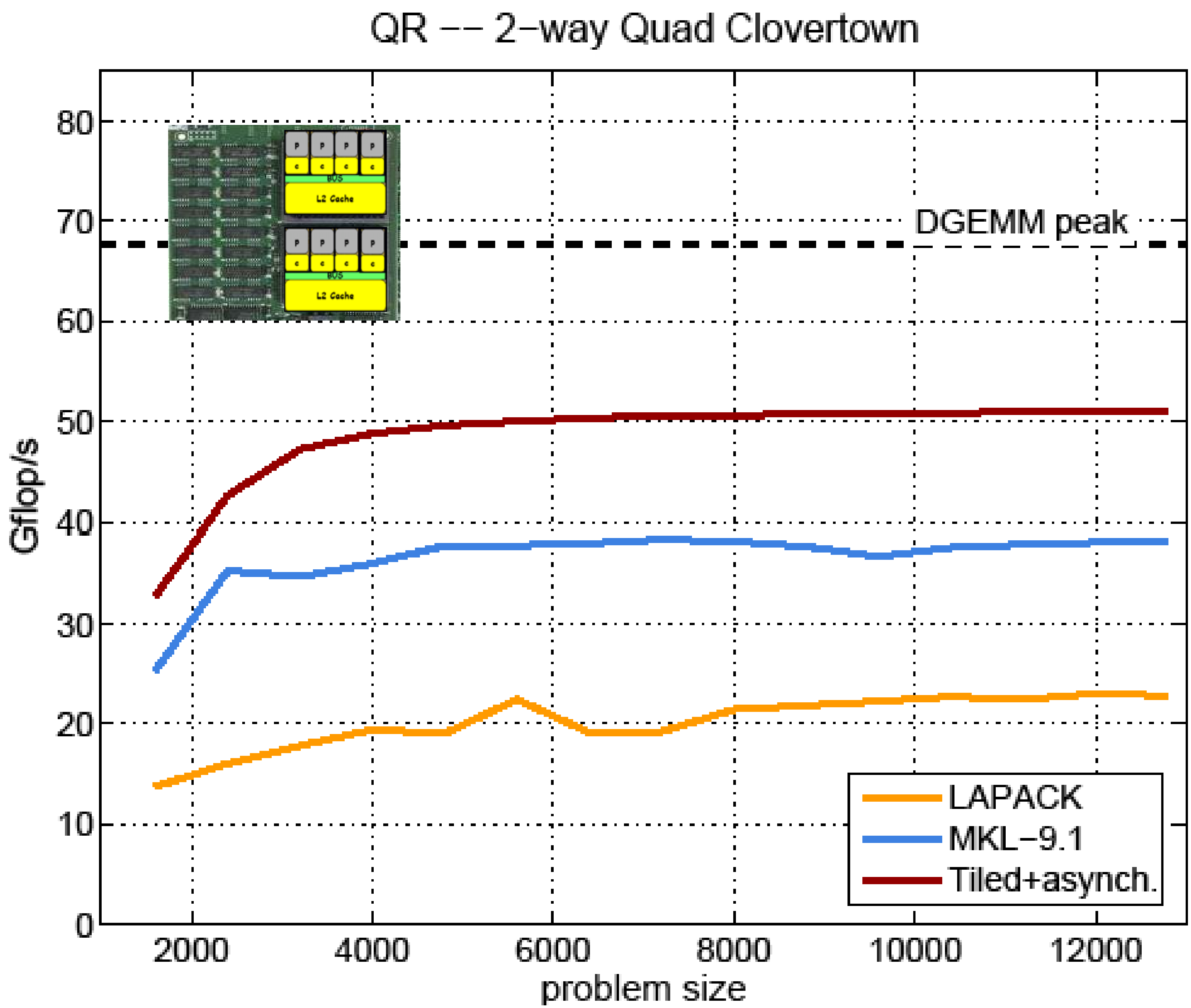
Blocked




LU -- 8-way dual Opteron -- MKL-9.1
Gflop/s
0
10
20
30
40
50
60
70
2000
4000
6000
8000
10000
12000
problem size
LAPACK
MKL-9.1
Tiled+asynch.
L2 Cache
BUS

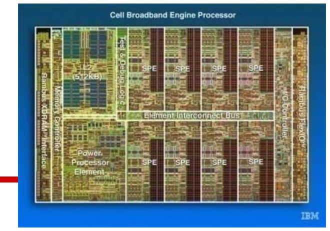


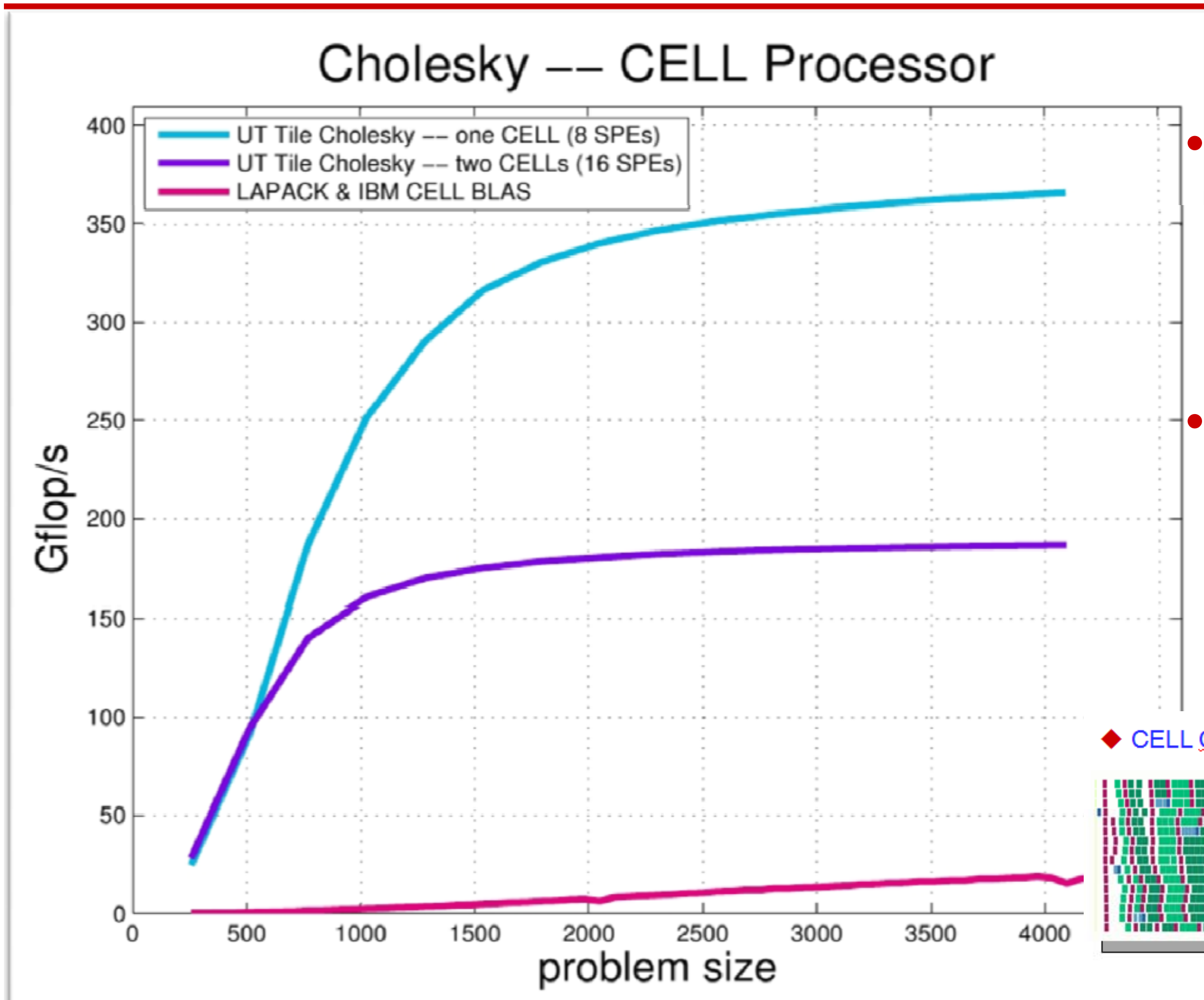
Cholesky -- 2−way Quad Clovertown
Gflop/s
problem size
LAPACK
MKL−9.1
Tiled+asynch.
DGEMM peak
L2 Cache
BUS
0
10
20
30
40
50
60
70
80
2000
4000
6000
8000
10000
12000



QR -- 2-way Quad Clovertown
DGEMM peak
L2 Cache
L2 Cache
BUS
BUS
Gflop/s
problem size
80
70
60
50
40
30
20
10
0
2000
4000
6000
8000
10000
12000
LAPACK
MKL−9.1
Tiled+asynch.

Cholesky on the CELL

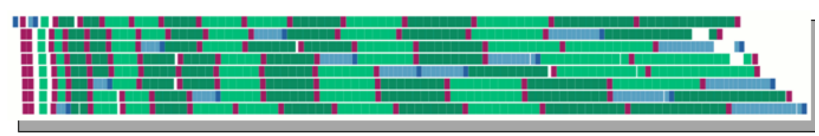




- 1 CELL (8 SPEs)
  - 186 Gflop/s
  - 91 % peak
  - 97 % SGEMM peak
- 2 CELLs (16 SPEs)
  - 365 Gflop/s
  - 89 % peak
  - 95 % SGEMM peak

- CELL Cholesky – 16 cores

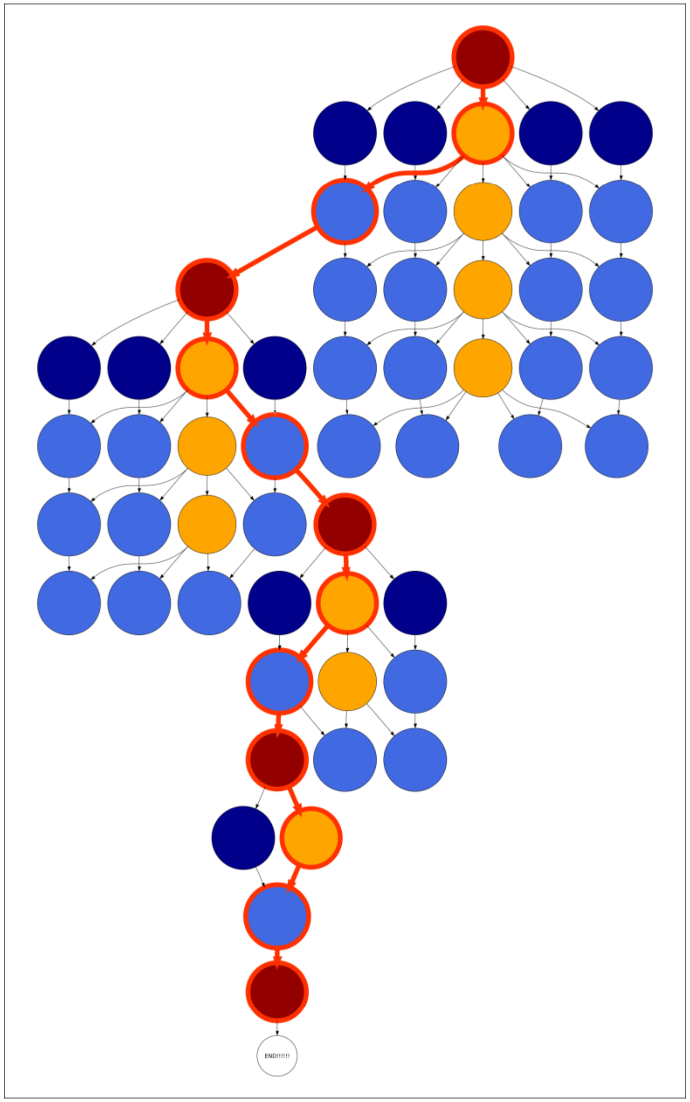
- CELL Cholesky – 8 cores



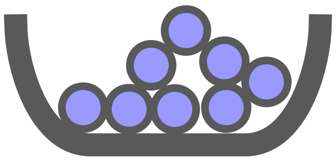
Single precision results on the Cell

If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system

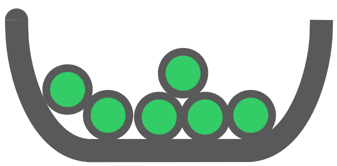


Each Node or Core Will Have



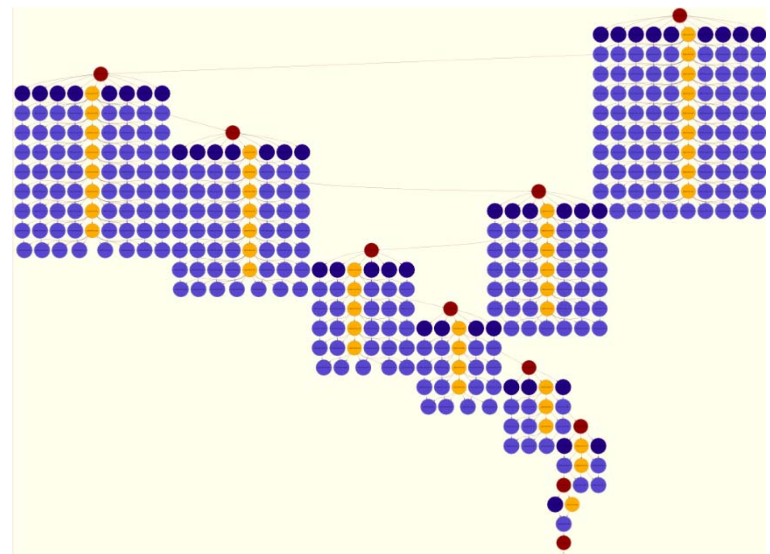
BIN 1

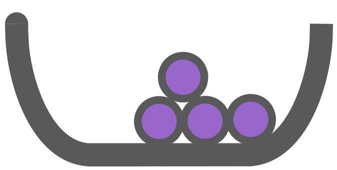
- some dependencies satisfied
- waiting for all dependencies



BIN 2

- all dependencies satisfied
- some data delivered
- waiting for all data





BIN 3

- all data delivered
- waiting for execution

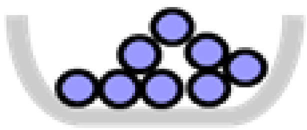
DAG and Runtime

Bin 1: Waiting for dependencies to be satisfied

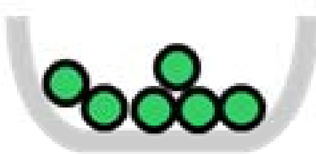
Bin 2: All dependencies satisfied, waiting for data

Bin 3: dependencies and data available; ready to execute

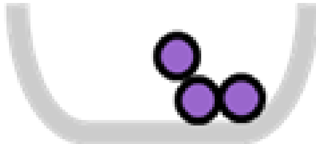
- **Execute task in Bin 3**
  - Task with all data and dependencies satisfied
  - After execution report to children done and dependencies satisfied and send data
  - Steal task if none
- **Check Bin 1 to see if new dependencies satisfied for tasks**
  - If new dependency satisfied update and post receive of data
  - If all dependencies and data available satisfied move to Bin 2
- **Check Bin 2 to see data arrival**
  - Check for data arrival; If all data available move to Bin 3
- **If needed place new task from my part of the DAG into Bin 1**



1 Some dependencies satisfied; waiting for all



2 All dependencies satisfied; waiting for data



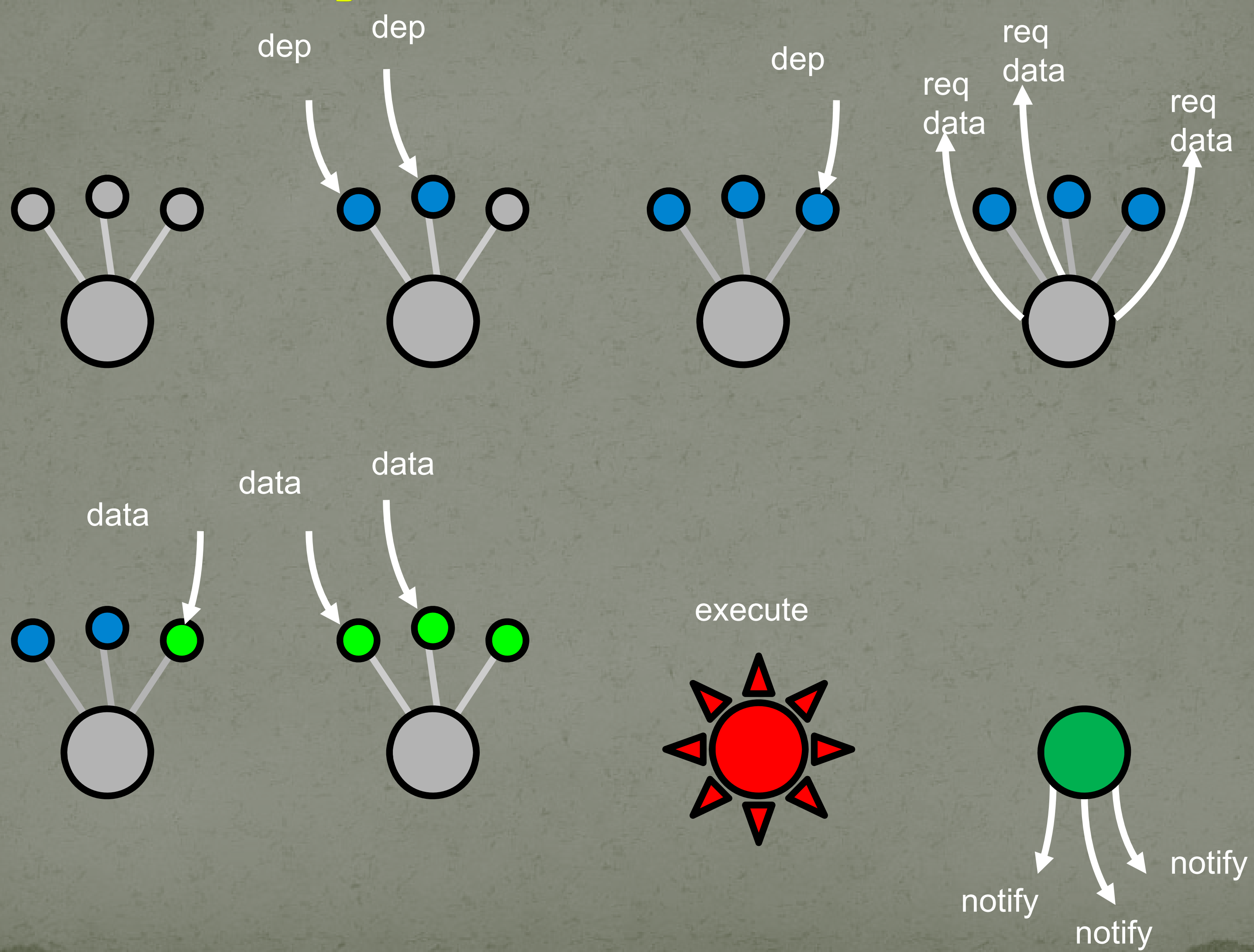
3 Waiting for execution

DAG and Scheduling

- **DAG is dynamically generated and implicit**
- **Everything designed for distributed memory systems**
- **Runtime system on each node or core**

- **Run time**
- **Bin 1**
  - **Exec a task that's ready**
  - **Notify children of completion**
  - **Send data to children**
  - **If no work do work stealing**
- **Bin 2**
  - **See if new dependences are satisfied**
  - **If so move task to Bin 3**
- **Bin 3**
  - **See if new data has arrived**

Task Life-Cycle



Looking Into a Number of Things

- **DAG must be dynamic**
  - **Some of the algorithms are iterative i.e. eigenvalue problem**
- **Parameterized Task Graph**
  - **Cosnard and Jeannot**
- **DAG has to be handled in a distributed fashion**

Some Questions

- What's the best way to represent the DAG?
- What's the best approach to dynamically generating the DAG?
- What run time system should we use?
  - We will probably build something that we would target to the underlying system's RTS.
- What about work stealing?
  - Can we do better than nearest neighbor work stealing?
- What does the program look like?
  - Experimented with Cilk, Charm++, UPC, Intel Threads
  - I would like to reuse as much of the existing software as possible

Collaborators / Support

Alfredo Buttari,
ENS/INRIA

Julien Langou,
U Colorado, Denver

Julie Langou,
UTK

Piotr Luszczek,
MathWorks

Jakub Kurzak,
UTK

Stan Tomov,
UTK