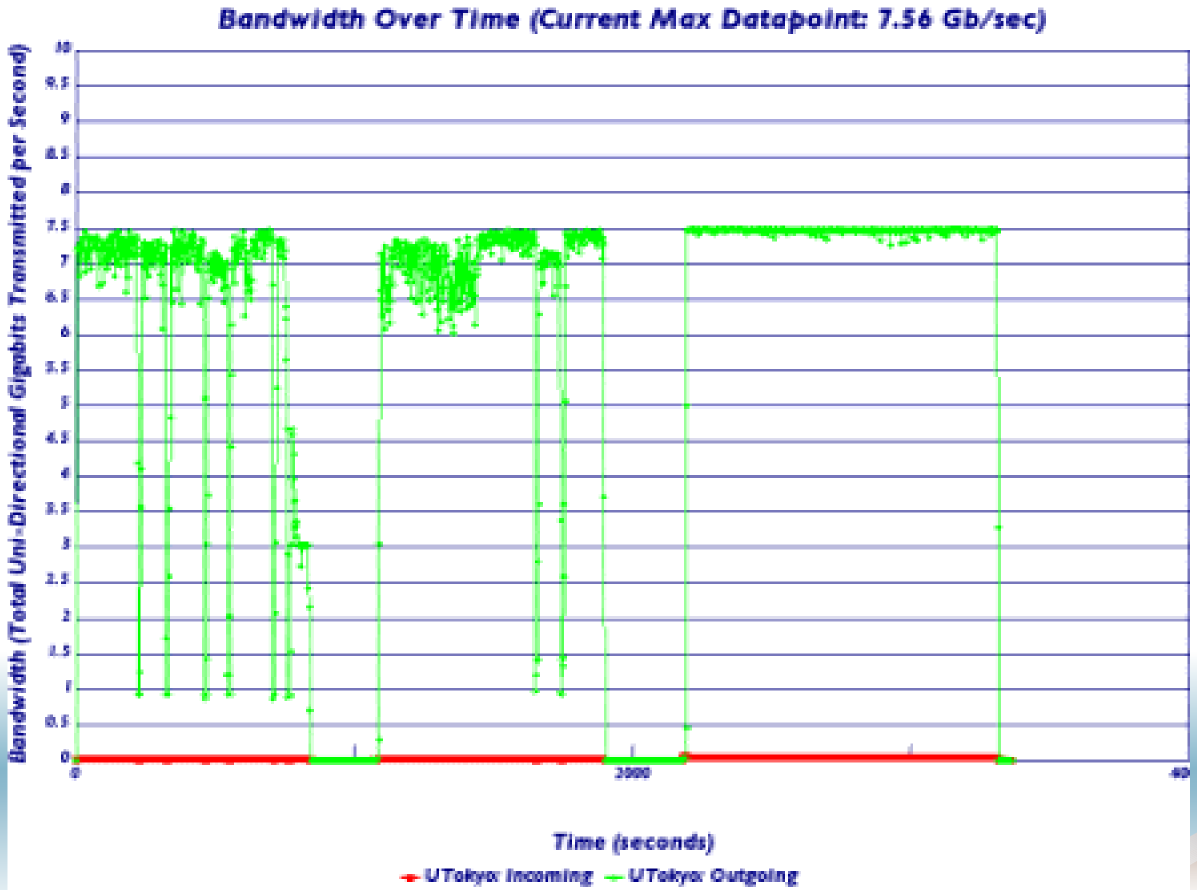
High Performance Computing on Grid

Kenjiro Taura
Toshio Endo, Kenji Kaneda
University of Tokyo

Long fat pipes are now real

- SC2003 Bandwidth Challenge

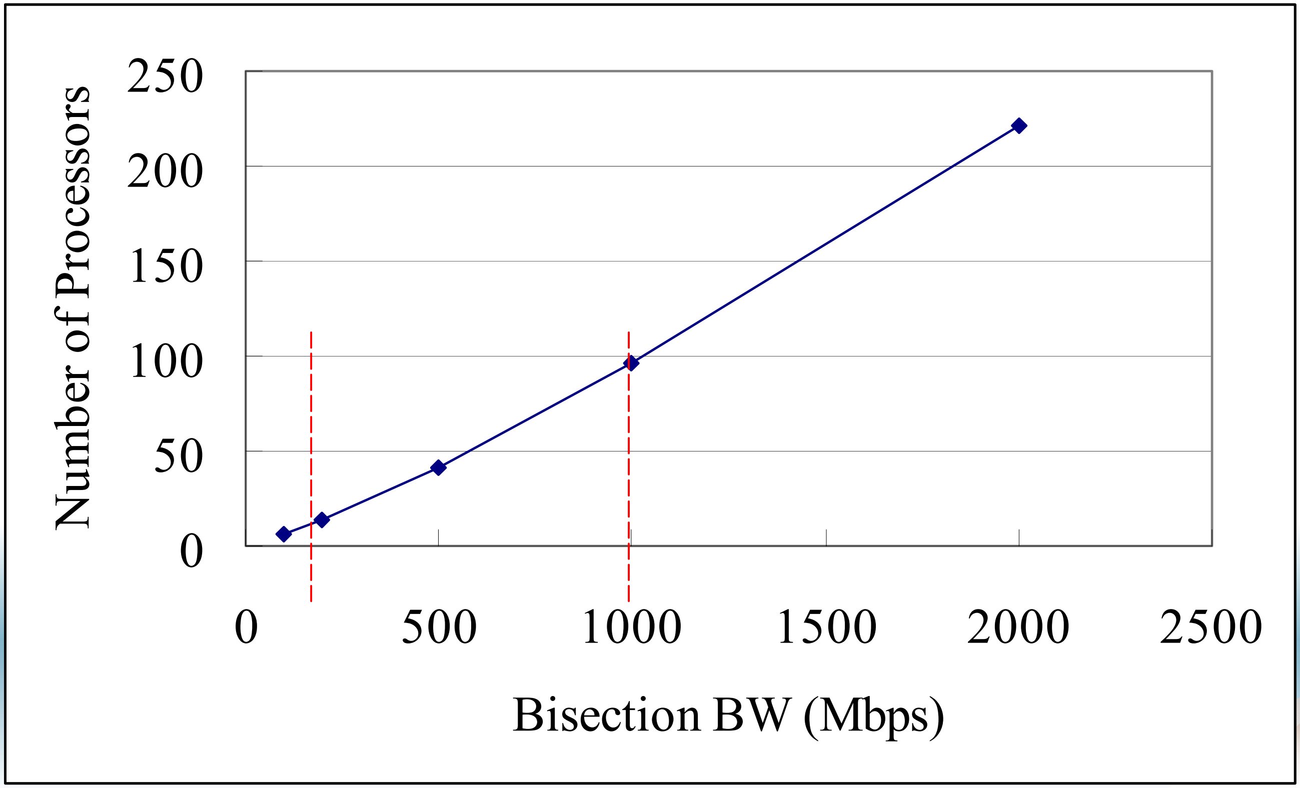


Implication

- WAN bandwidth limitations will be gone
- Today's common conception must become misconception
  - Grid only for brute-force, task-farming APPs
  - Traditional HPCs only on clusters

Analyzing LU scalability

- “Bisection BW” vs “Number of procs comfortably supported” (1 min. jobs)



Our question

- Are “fat pipes” all that’s required to make “HPC on Grid” real?
- If not, what else are required?

What's in today's Grid?

- Data sharing
- Small scale coordination across clusters
- Brute-force & compute-mostly tasks

What's *not* in today's Grid?

- Sense of "an alternative to a cluster for HPC"
  - "my fluid dynamic code would be too slow"
- Sense of "resources pooled and shared *on demand*"
  - "cluster X is a bit occupied, so I'll rather wait until tomorrow…"

Opportunities with fat pipes

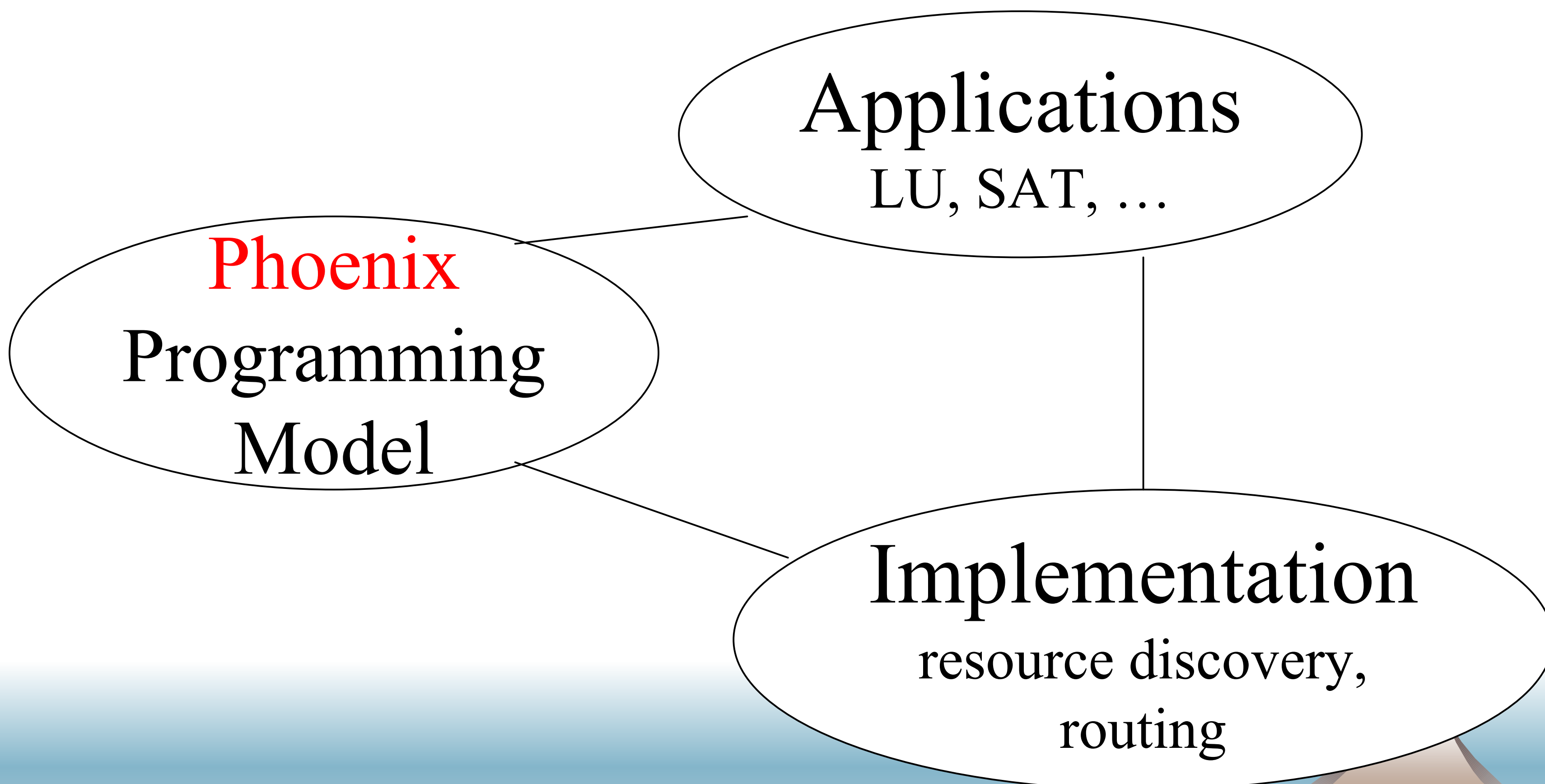
- Communication-intensive apps
- Dynamically/automatically chosen resources
- Migration of app states to reconfigure resources at runtime

They all contribute to make Grid close to what it ought to be

Are fat pipes *all* that's required?

- Obviously no
- We need a range of research to make the above story real
- Key issue
  - make communication-intensive HPC apps *flexible, adaptable, and latency-tolerant*

Our research theme and approach

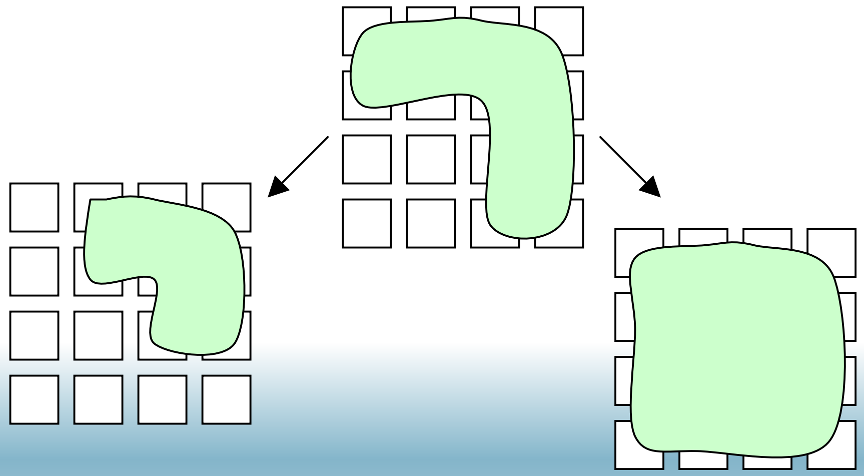


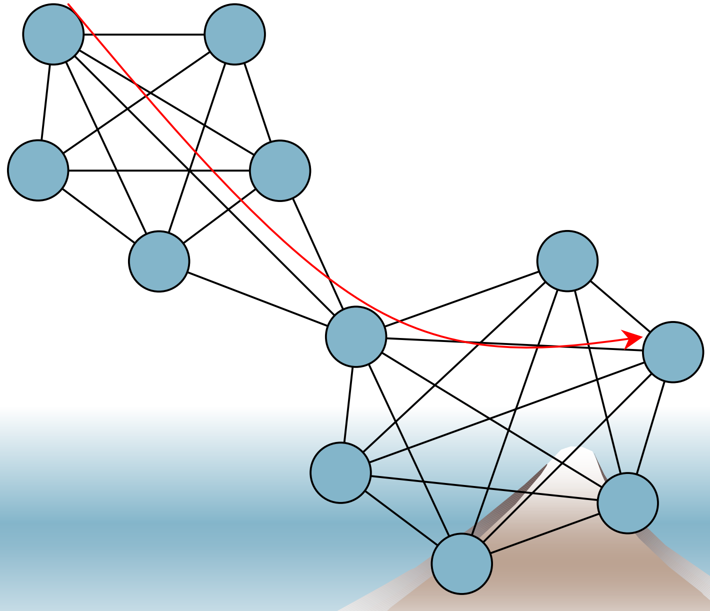
Phoenix programming model

- Baesd on *general* message passing model
- Unlike usual message passing models, however, it is designed on the assumption that *nodes are selected, added, or deleted at runtime* (by user, scheduler, etc.)

Phoenix *key* features

- *Transparent migration of app state*
- *Transparent communication over WAN (with firewalls, NAT, DHCP, etc.)*





Why important?

- It allows participating nodes
  1. to change over time, to adapt to dynamic conditions (host load, network traffic, etc.)
  2. to be flexibly selected by external agents (resource scheduler, broker, etc.)
  3. to be dead in the beginning
  4. to crash at runtime (with suitable provision in app logic)

To summarize,

fat pipes + Phoenix

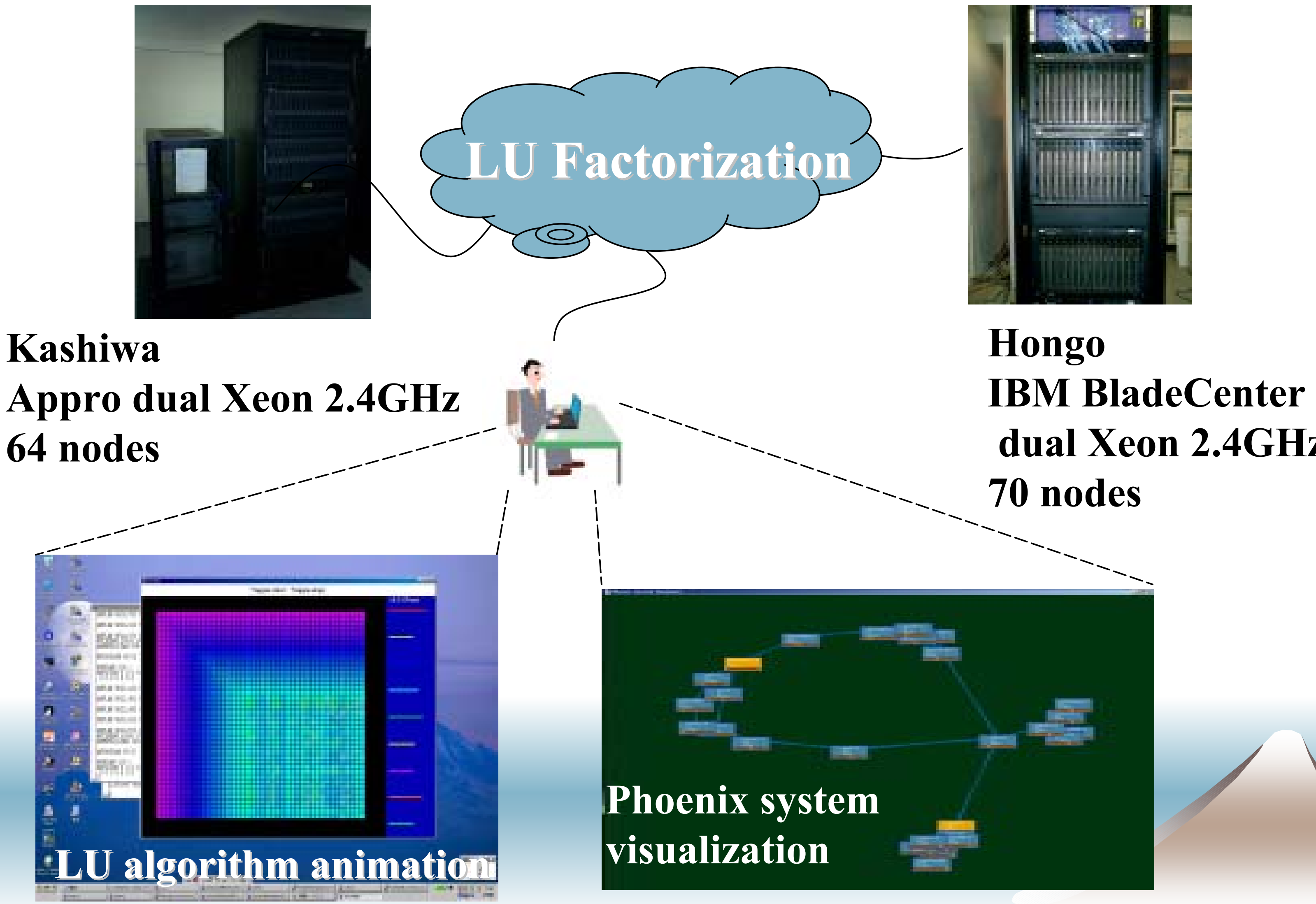
= HPC with dynamic resources. adaptive, fault tolerant, etc.

= Grid as a real pool of resources

Our experience in LU factorization

- Endo et al. [CCGrid 2004]
- *Asynchronous* LU factorization written in Phoenix
  - runs over multiple LANs
  - tolerates background loads and long latencies
  - allows nodes to be added at runtime

Phoenix Demo



LU Factorization Basics

- for (k=0; $k<N$; k++) {

  for ($j=k$; $j<N$; j++) $A_{kj} = A_{kj} / A_{kk}$;

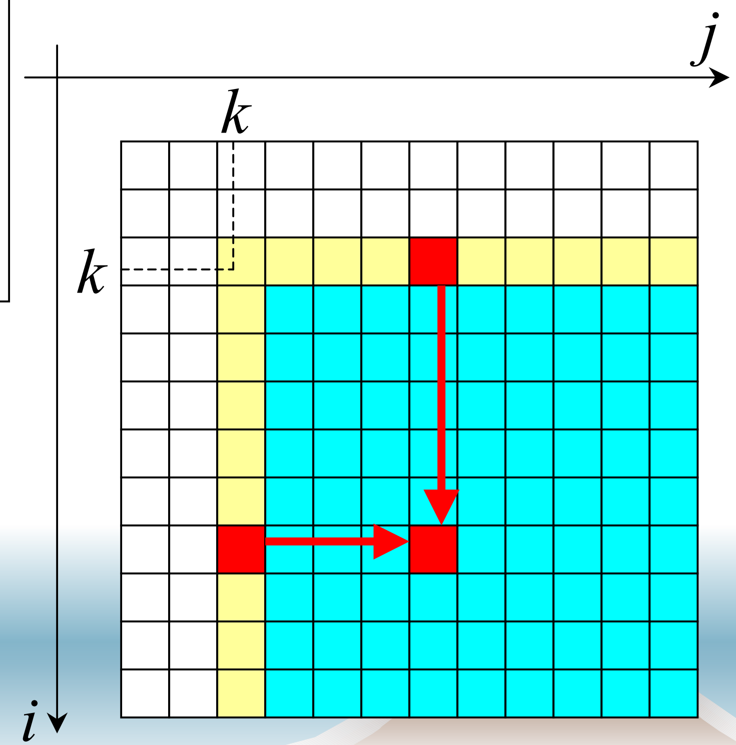
  for ($i=k+1$; $i<N$; i++)

  for ($j=k+1$; $j<N$; j++)

  A_{ij} −= $A_{ik} A_{kj}$;

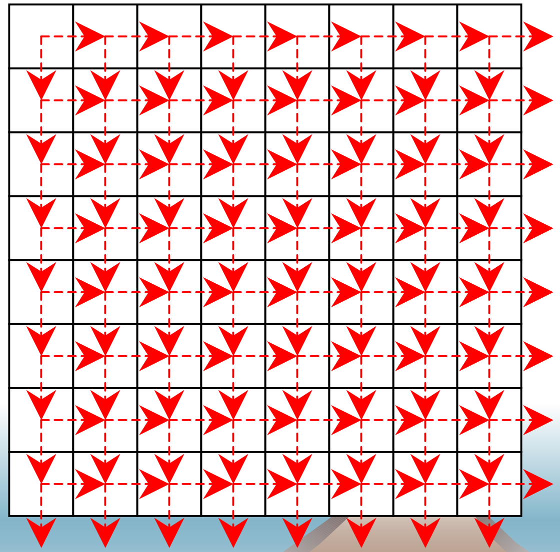
  }

update

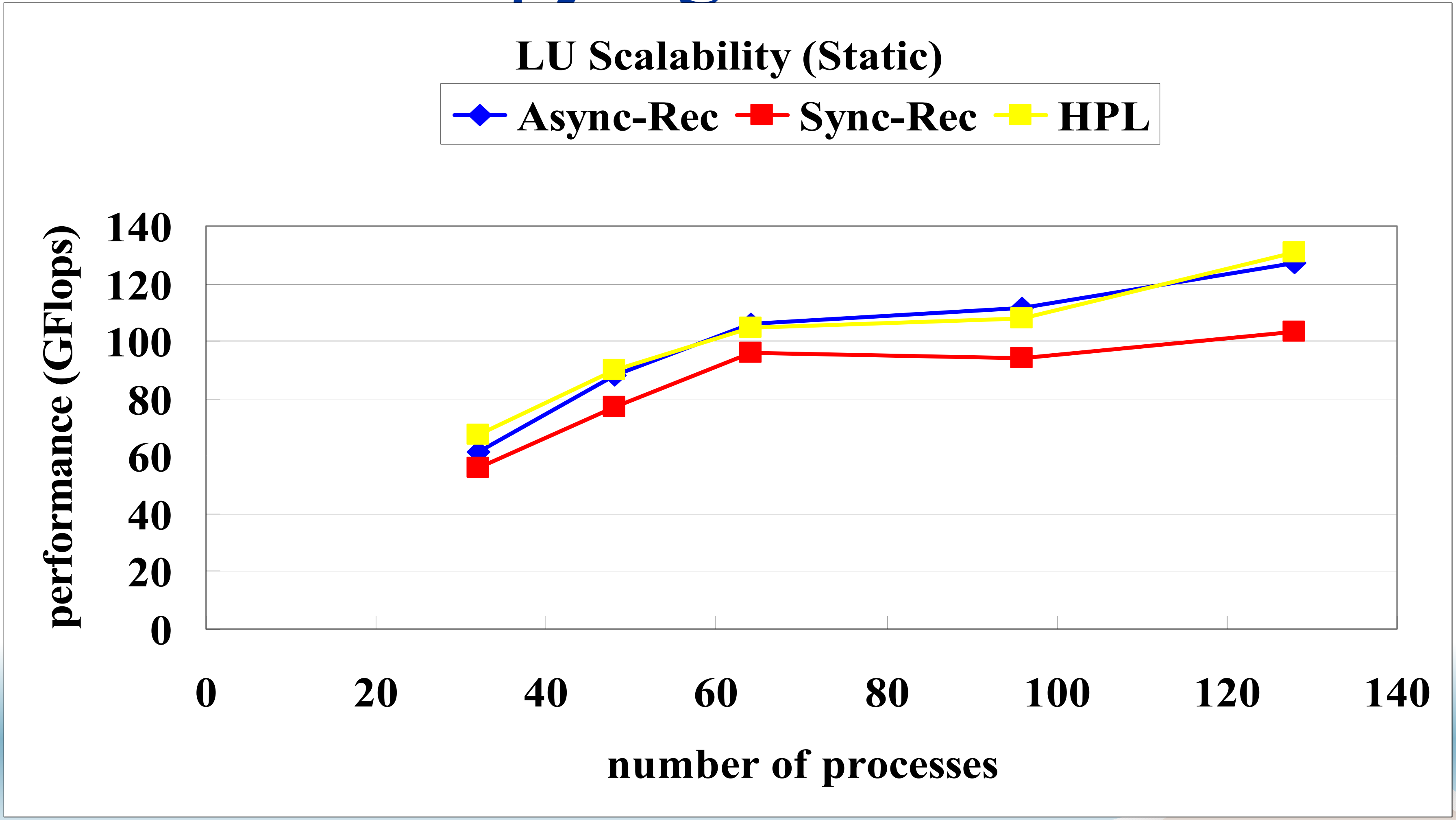


Asynchronous LU Factorization

- Computation on a block fires as soon as necessary data arrives
- multiple *k* iterations overlap
  - latency tolerant
  - background load resilient



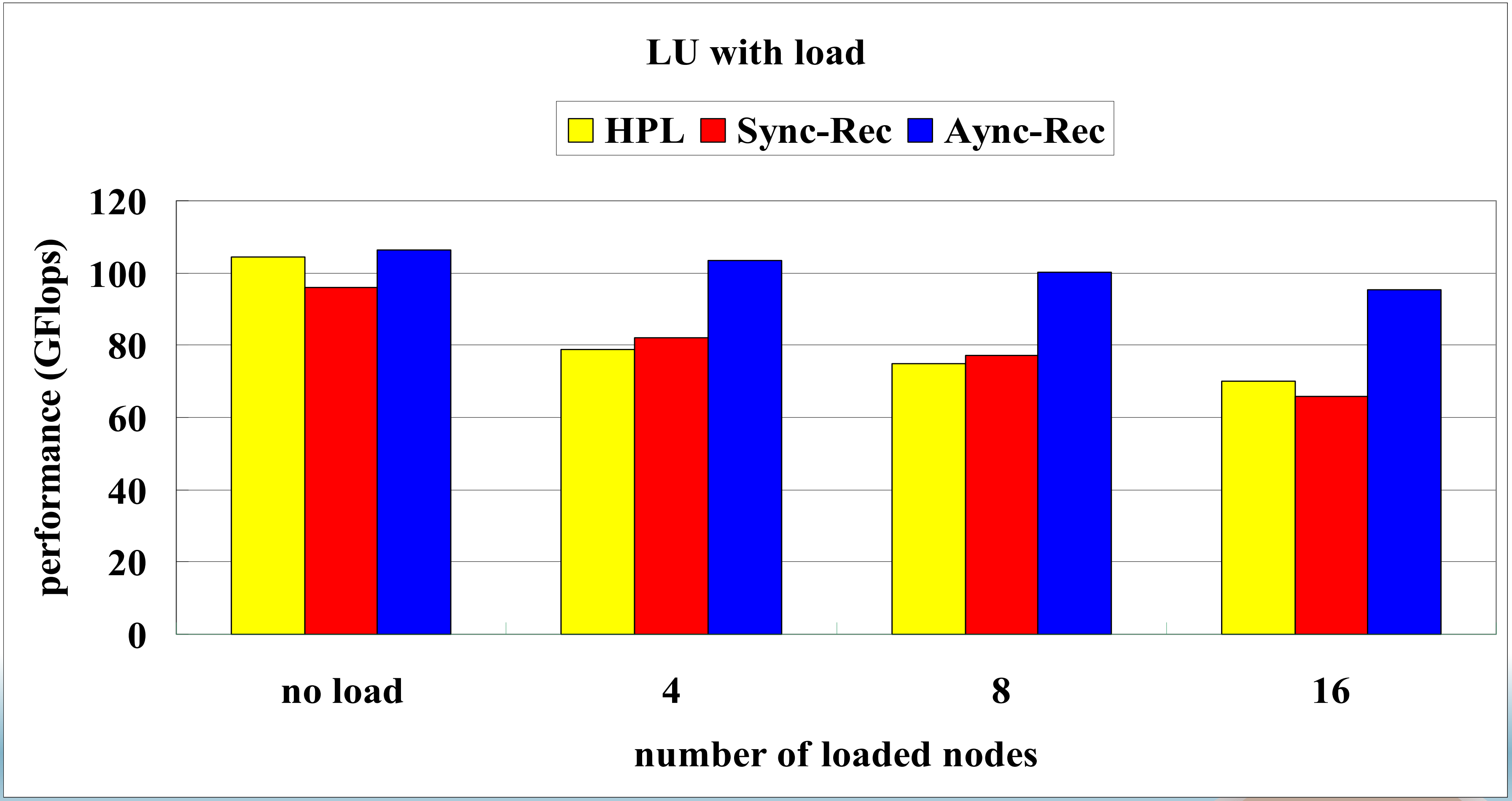
Performance when exclusively occupying the cluster



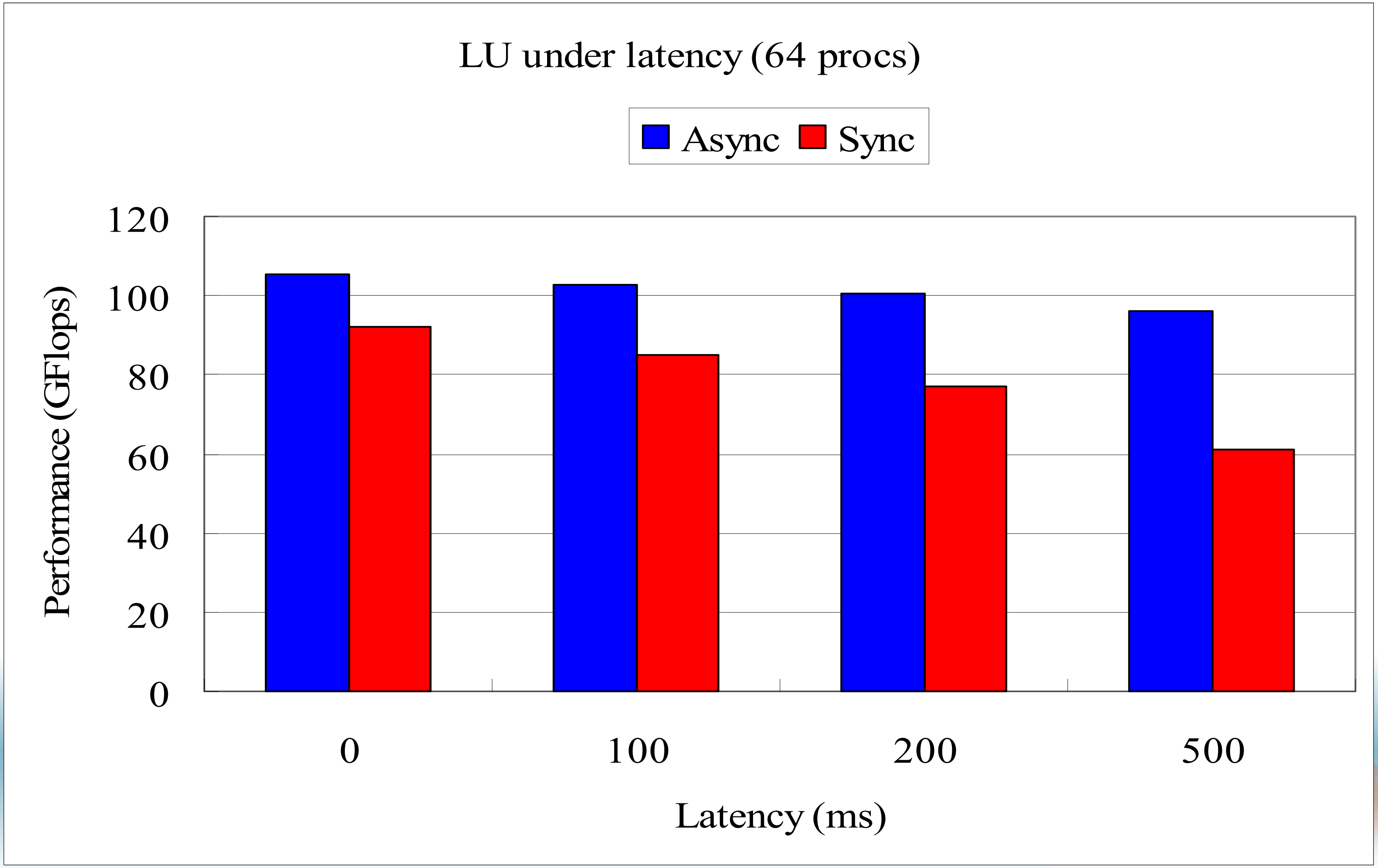
A reference data

- From 21st Top 500 (Xeon 2.6GHz 150CPU, GigE)
  - N=126,000 ⇒ 365Gflops
  - N=50,000 ⇒ 182.5Gflops
- Comparison to HPL (High Performance Linpack; MPI)
  - N = 25,000, 64 processes
  - HPL ⇒ 110Gflops
  - Phoenix ⇒ 108Gflops

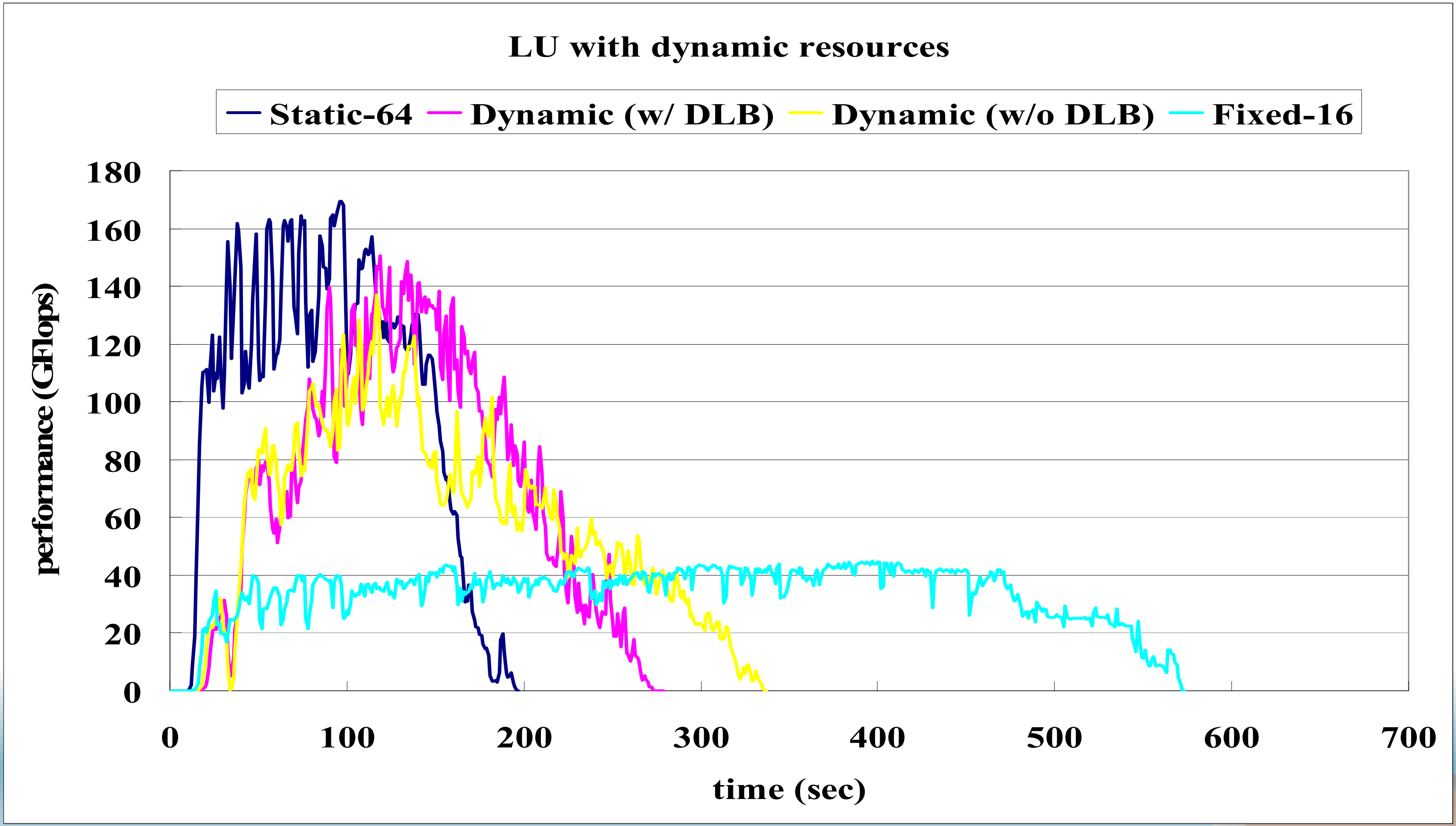
with background load



with large latencies



Adding processes dynamically



Conclusion

Wrong directions	Right directions
exclusively focus on peak performance	seriously investigate performance under various loads
stick to MPI + NPB	new models, both traditional and new apps
stick to single clusters for HPCs	make you HPC apps flexible
stay in “master-worker” models on the Grid	everything on the Grid