Network Centric Runtime Implementation

Costin Iancu
Khaled Ibrahim

The Stampede to Exascale

- **Hardware is changing to satisfy power constraints**
  - $O(10^3)$ cores per node/socket, $O(10^6)$ nodes
  - Nodes are hybrid, asymmetric
  - Network is under-provisioned (tapered = 0.1 bytes/flop, not fully connected)
- **New application domains are emerging**
  - Some regular, embarrassingly parallel (Bioinformatics)
  - Some are irregular, hard to parallelize (ExaCT)
- **Programming models are changing to reflect hardware and apps**
  - Shared memory, Global Address Spaces
  - Fine grained asynchrony – parcels, activities, fibers….
  - Unstructured parallelism – finish/async, phasers …

Integrator Needed

- **Intra-node programming is the CHALLENGE!**
  - Focus on shared memory programming
  - Multiple paradigms, projects, languages, runtimes
    - Data parallel (CUDA, OpenCL, OpenMP…)
    - Dynamic tasking (OpenMP, X10, Chapel, HPX, SWARM)
    - Work stealing (Habanero, X10, Cilk, Intel TBB)

Integrator Needed

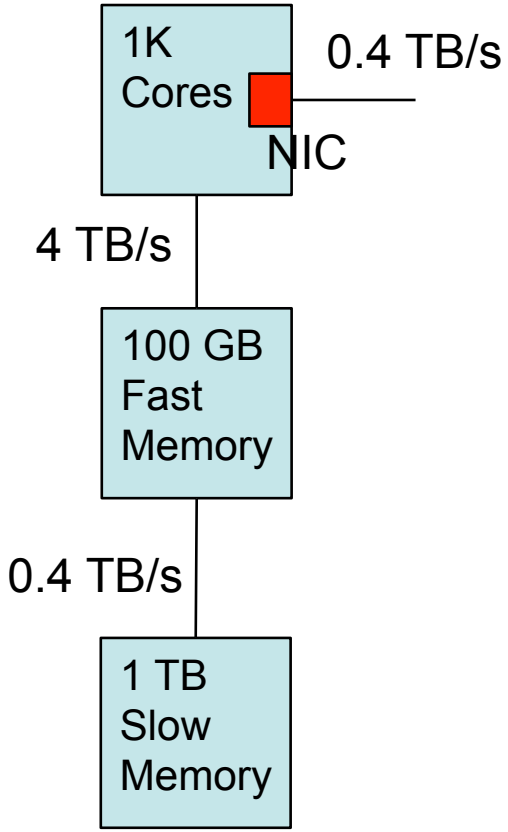
- **Intra-node programming is the CHALLENGE!**
  - Focus on shared memory programming
  - Multiple paradigms, projects, languages, runtimes
    - Data parallel (CUDA, OpenCL, OpenMP…)
    - Dynamic tasking (OpenMP, X10, Chapel, HPX, SWARM)
    - Work stealing (Habanero, X10, Cilk, Intel TBB)

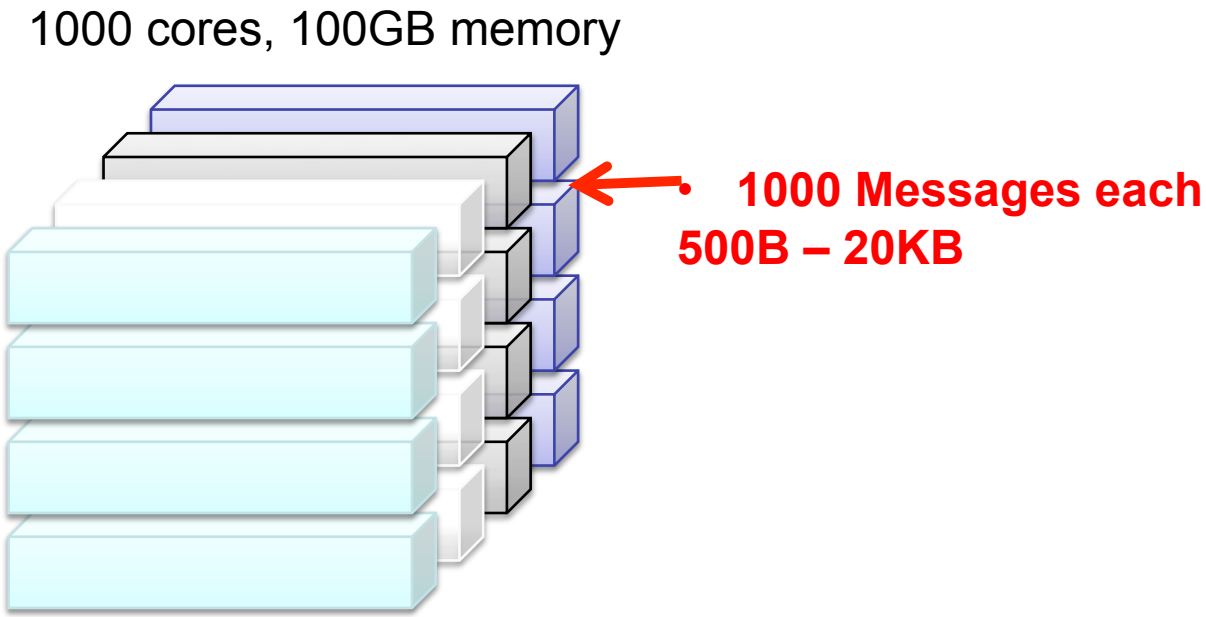
- **Integration with the network and whole system efficient utilization is another CHALLENGE!**
  - Node architecture/programming tackled by industry/academia/HPC
  - HPC networking is a niche market
  - Areas that require progress
    - System software support
    - Performance models
    - Dynamic optimizations

Integration Goals

- **Efficiency and productivity layer for heavily threaded/asynchronous applications**
  - **Productivity**
    - Decouple application/runtime level concurrency from runtime concurrency
    - Manage asynchrony for clients
  - **Performance portability = optimal throughput for**
    - Any implementation (pthreads, procs …)
    - Any hardware architecture (asymmetric, heterogeneous)
    - Any message mix
    - Any source, target

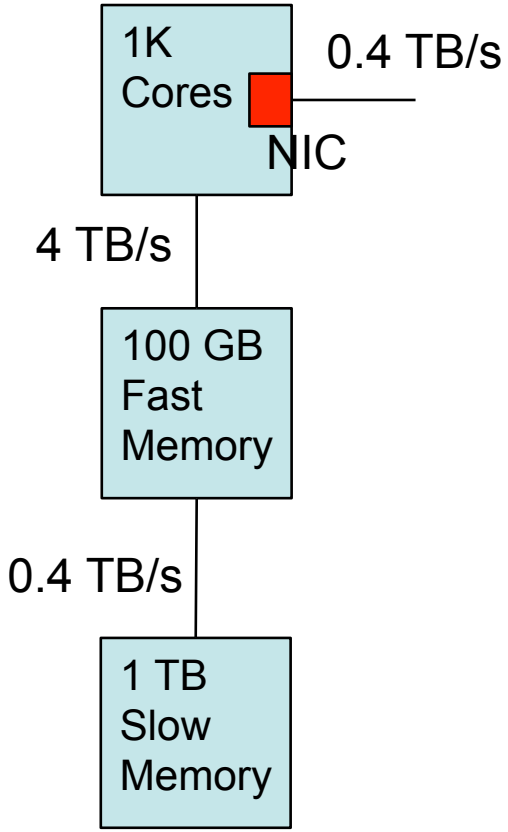
Optimization Goals



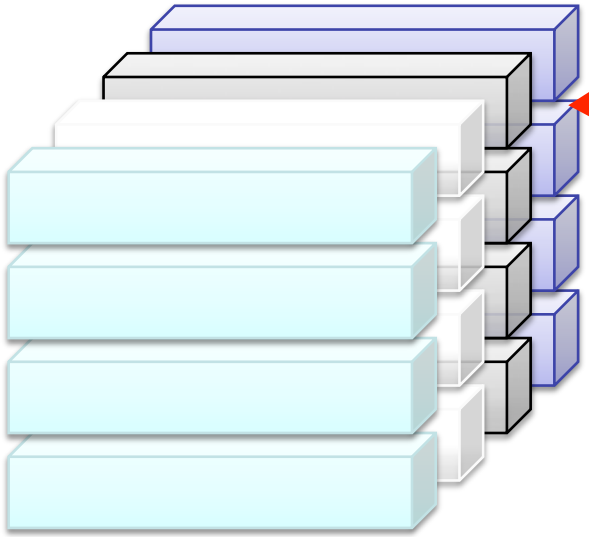


Small message throughput

Optimization Goals



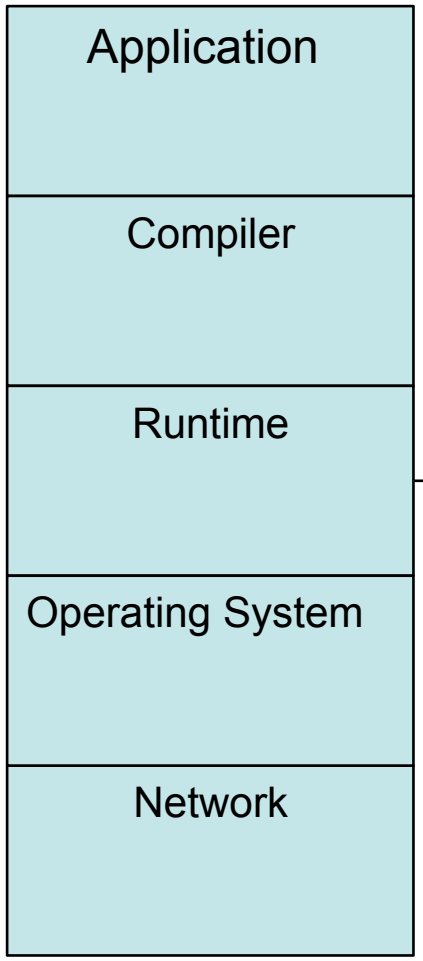
1000 cores, 100GB memory



- **1000 Messages each 500B – 20KB**
- **Aggregation**: 500KB message
  - Programmer
  - **Runtime**

Small message throughput AND Dynamic message coalescing + Large messages

Runtime Components



- Dynamic program analysis & representation
- Performance models
- Optimization methodology

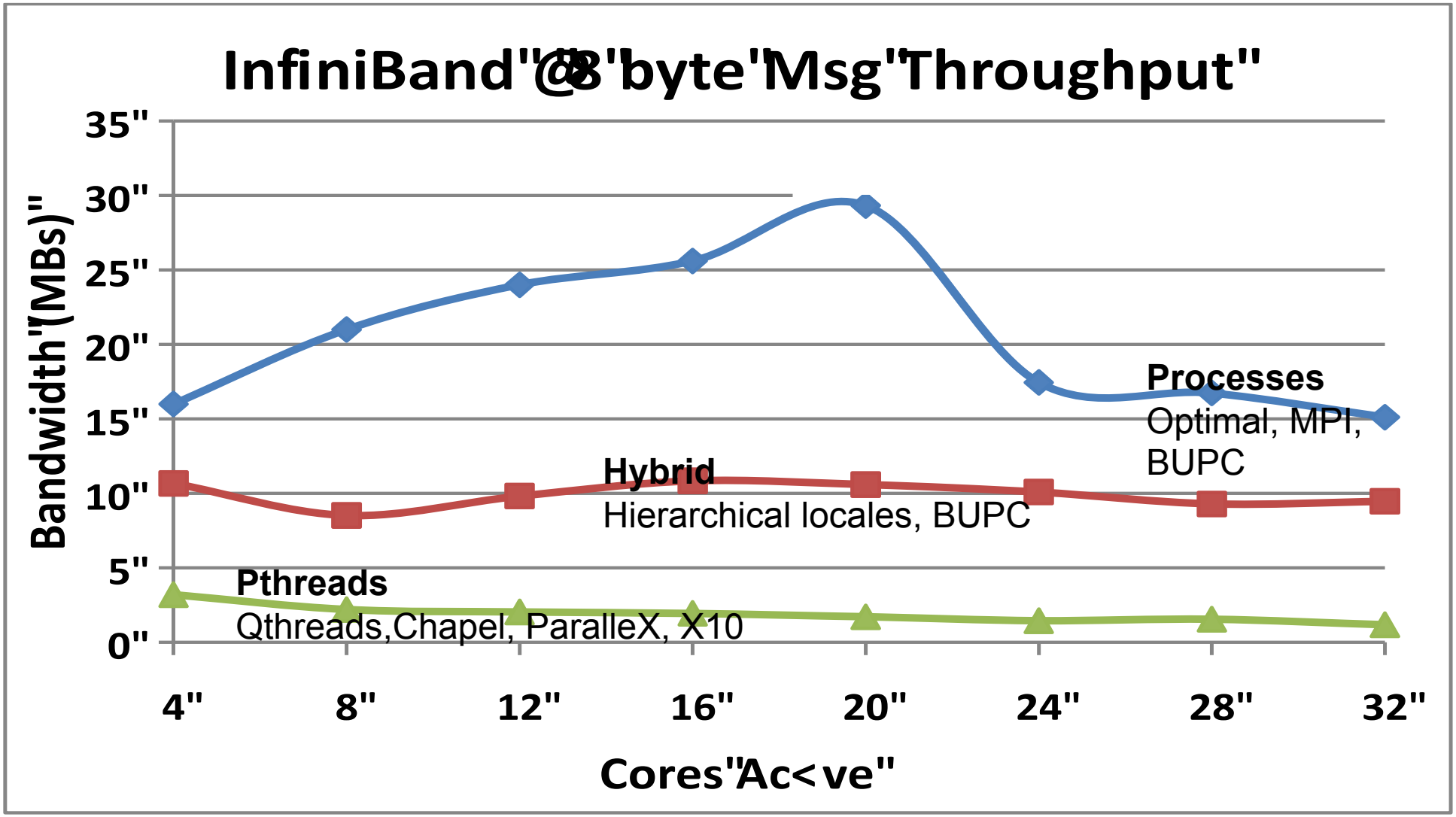
❖ **Dynamic communication optimizations**

❖ **Runtime Scheduler**

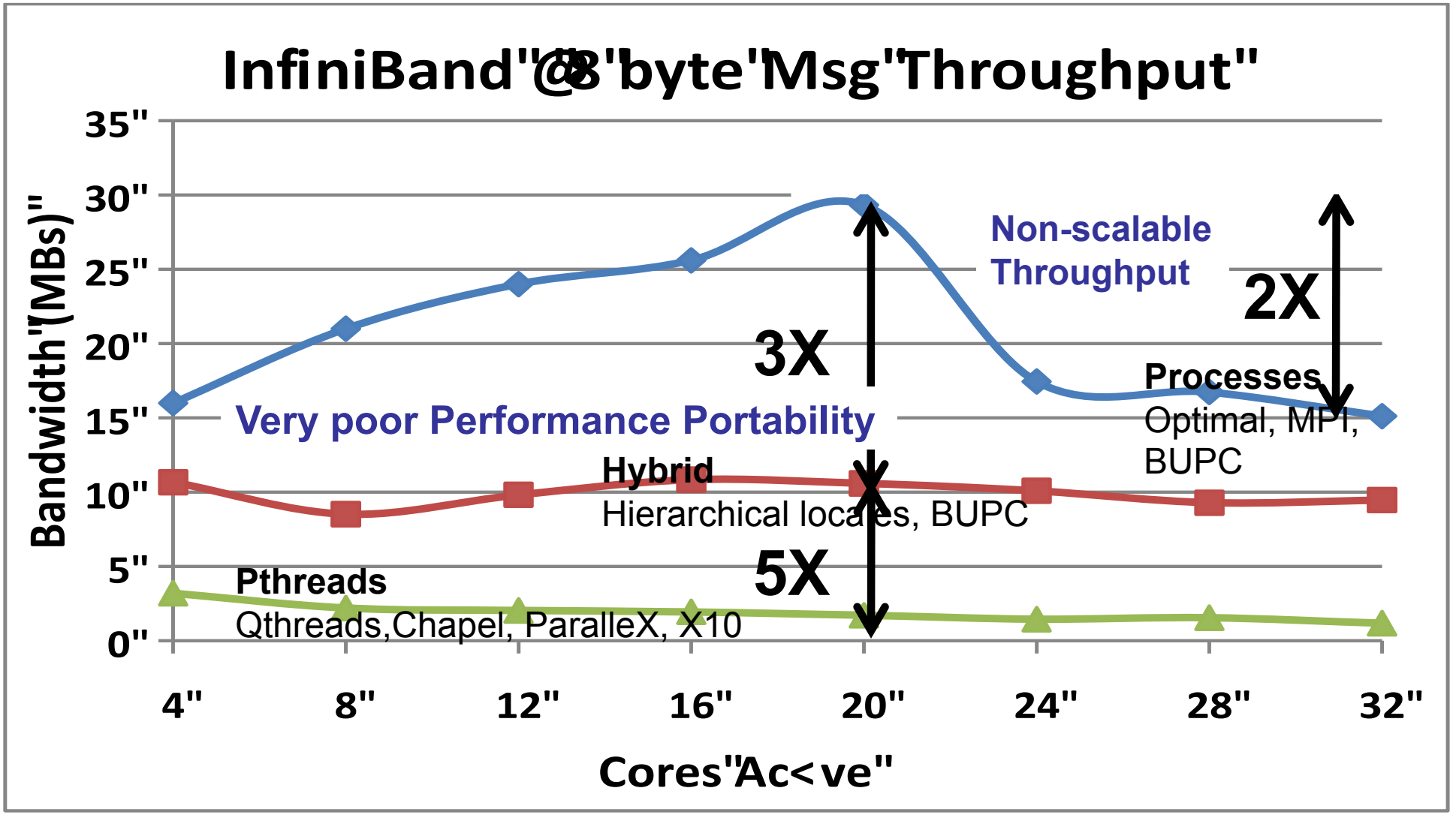
- Inject/retire independent messages
- Message re-ordering
- Match network concurrency with core concurrency
- Flow Control

Networks and Message Throughput

InfiniBand Performance



Performance: Implementation Matters!



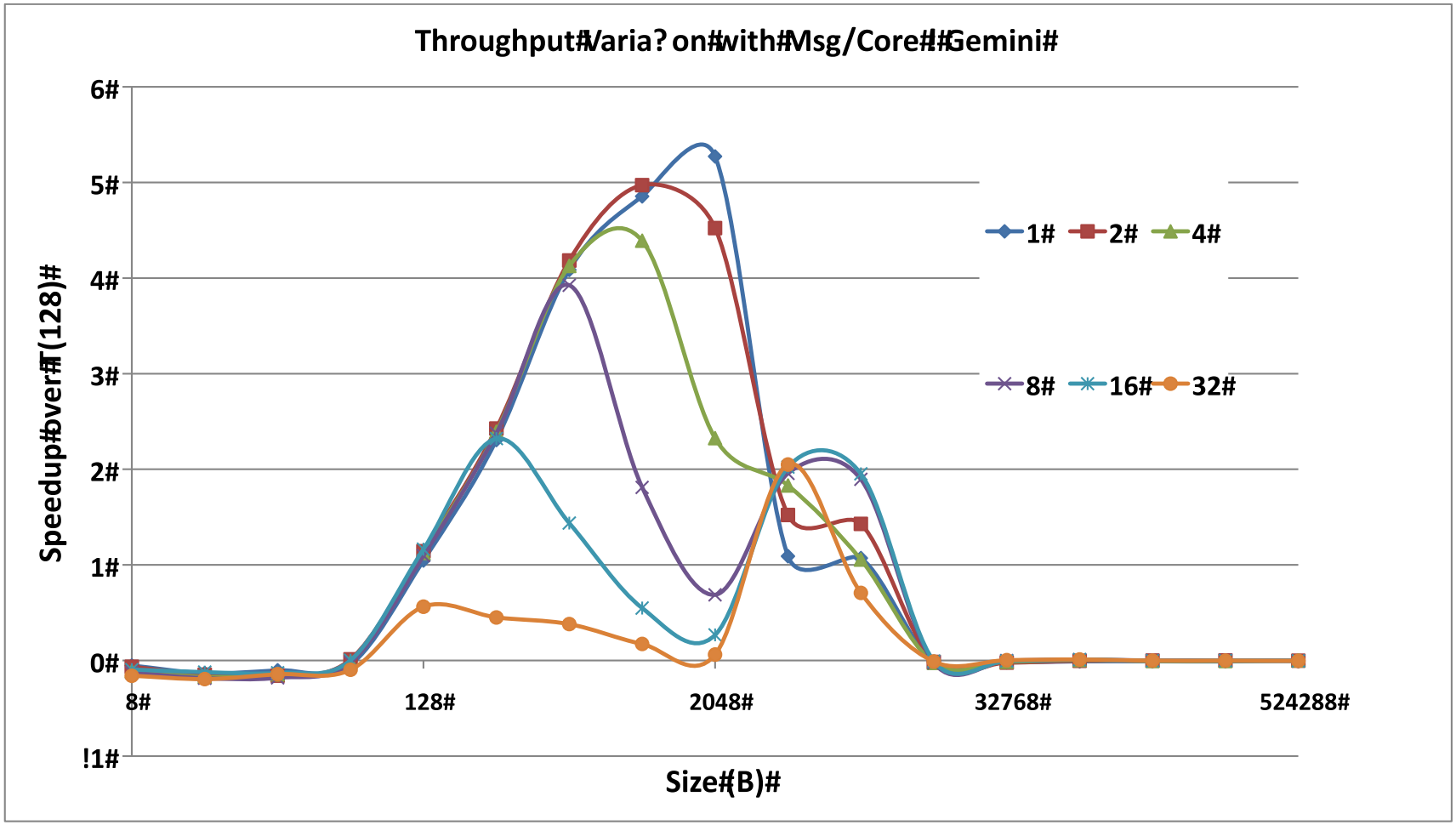
Throughput and Core Concurrency

BUPC/GASNet on InfiniBand

Throughput and Core Concurrency

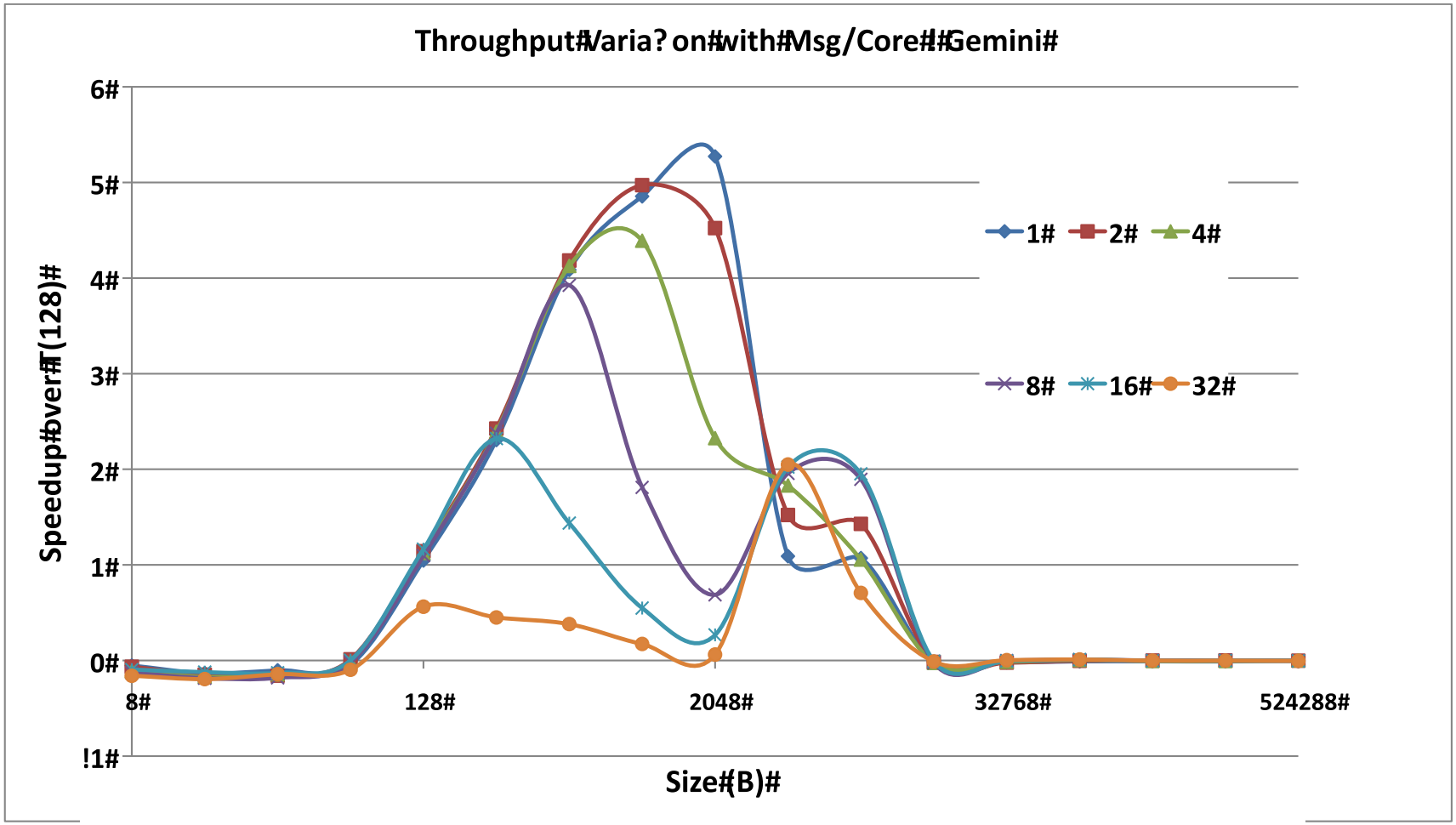
Serializing communication using 16 cores 40% faster than using 32 cores (expected 2x slower)

Throughput and Message Concurrency



Cray UPC on Cray XE6 (Gemini)

Throughput and Message Concurrency



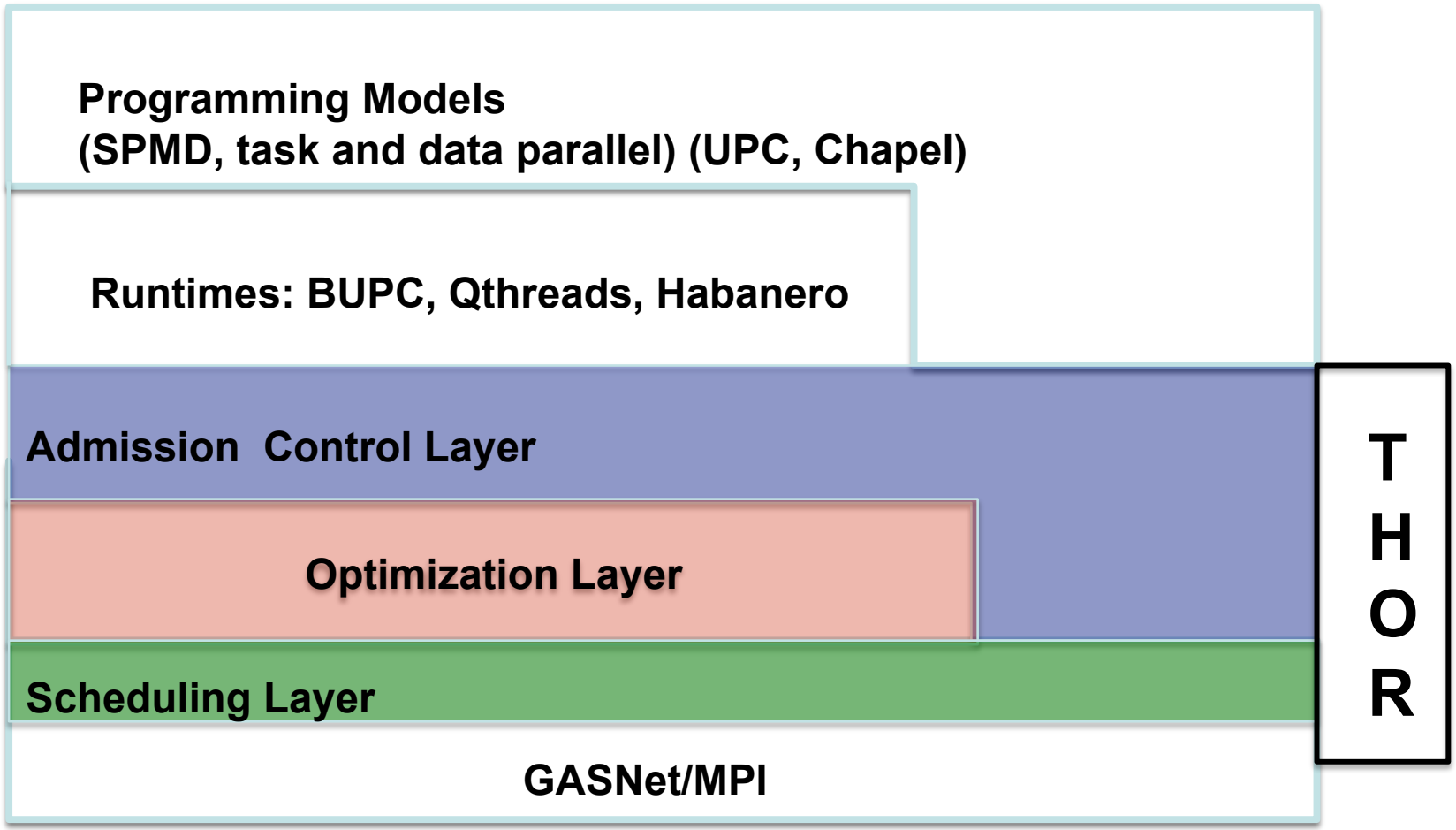
Limiting the number of outstanding messages provides 5X speedup (expected 32X slower)

Throughput Oriented Runtime for Large Scale Manycore Systems (THOR)

New Performance Metrics

- **Current runtimes optimized for single core latency and bandwidth**
  - Design and implementation
  - Micro-benchmarks and evaluation
- **I want to optimize for throughput**
  - Benchmarks
  - Metrics – is it msgs/sec or need delay guarantees too?
- **Analytical model**
  - Have talked before about LogGP for multicore
  - Or just empirical?
  - Is there a roofline?

Software Architecture



Driven by runtime analysis and performance models

Thor Layers

- **Admission Control Layer**
  - Congestion Avoidance
  - Flow Control
  - Concurrency matching
  - Memory Consistency/Ordering
  - Dispatch to Optimization Services
- **Optimization Layer**
  - Coalescing
  - Aggregation
  - Reordering
- **Scheduling Layer**
  - Integrate communication with tasking
  - Instantiate and Retire Communication to Network

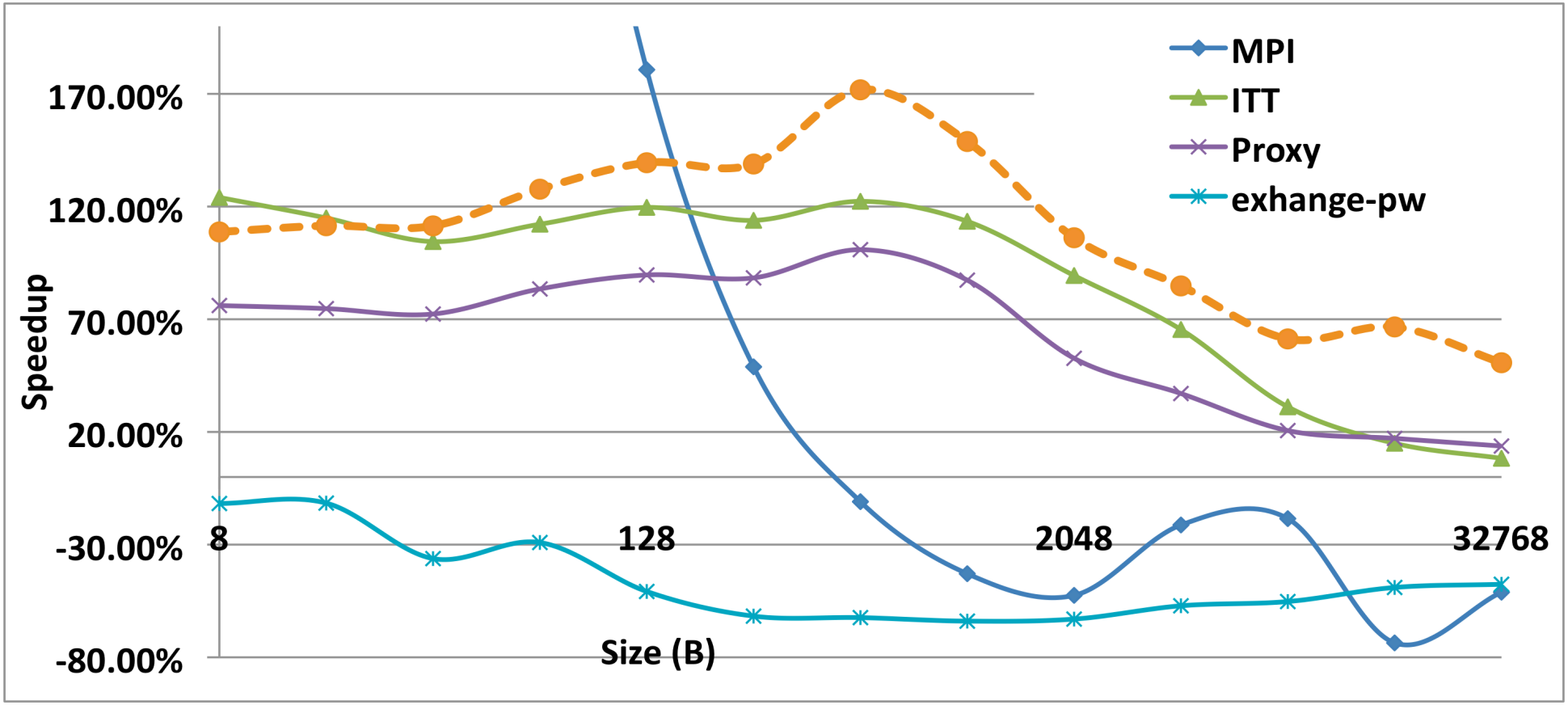
Congestion Avoidance

- **Throttle traffic to OPTIMAL concurrency**
  - Use micro-benchmarks to explore space
- **Proactive Management instructed by Declarative Behavior**
  - Catalogue of known "patterns"
  - Intuitive descriptions (e.g. all2all), annotated by compilers/humans
- **Integrated communication and task scheduling**
  - Inline: mechanisms implemented in a distributed manner
  - Proxy: servers acting on behalf of clients
- **Open loop control for scalability**
  - With as little "global" state as possible

Initial Results

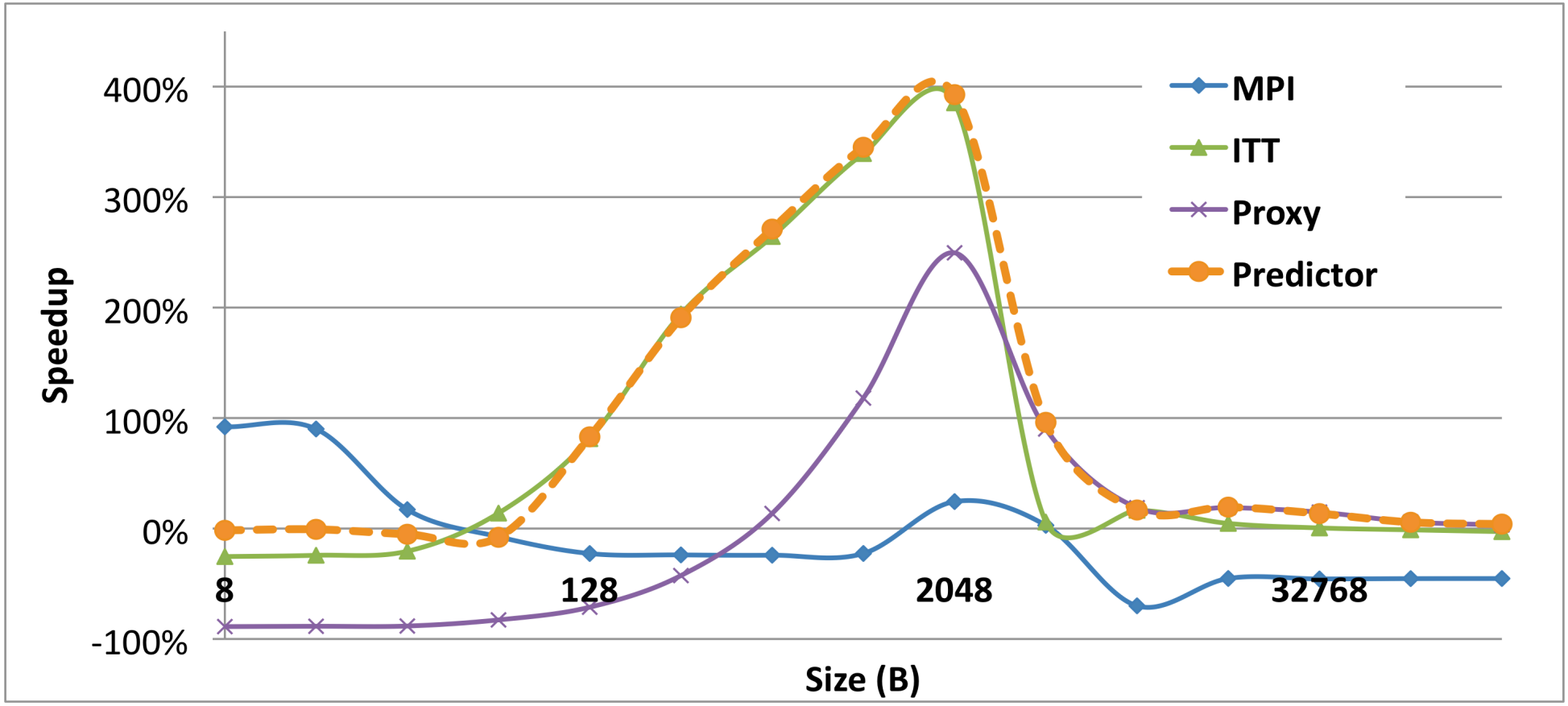
- **Prototype**
  - BUPC/GASNet/InfiniBand
  - Cray UPC/DMAPP/Gemini
- **Admission Control + Scheduling Layer**
  - Not well tuned yet
- **Results:**
  - **4X** performance improvement for all-to-all
  - **70%** improvement on GUPS/HPCC RA
  - **17%** on NAS Parallel Benchmarks

All-to-all InfiniBand 1024 Cores



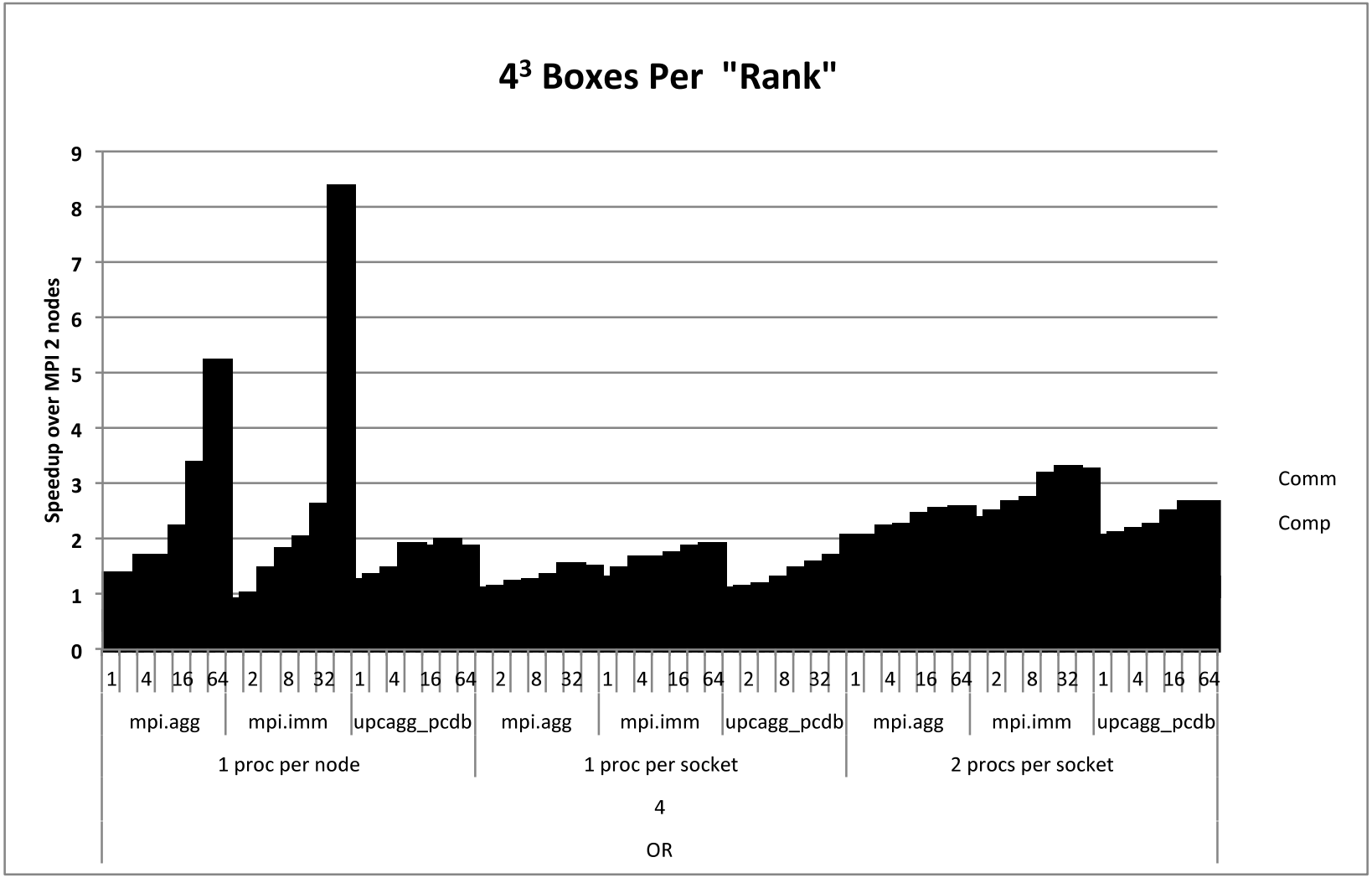
Speedup over GASNet tuned all-to-all - 2x Performance Portable – single implementation

All-to-all Gemini 768 Cores



Speedup over Cray UPC all-to-all - 4x
Performance Portable – single implementation

Plans/Projects



Dynamic Communication Optimizations

Case Study: Performance Portable Message Vectorization

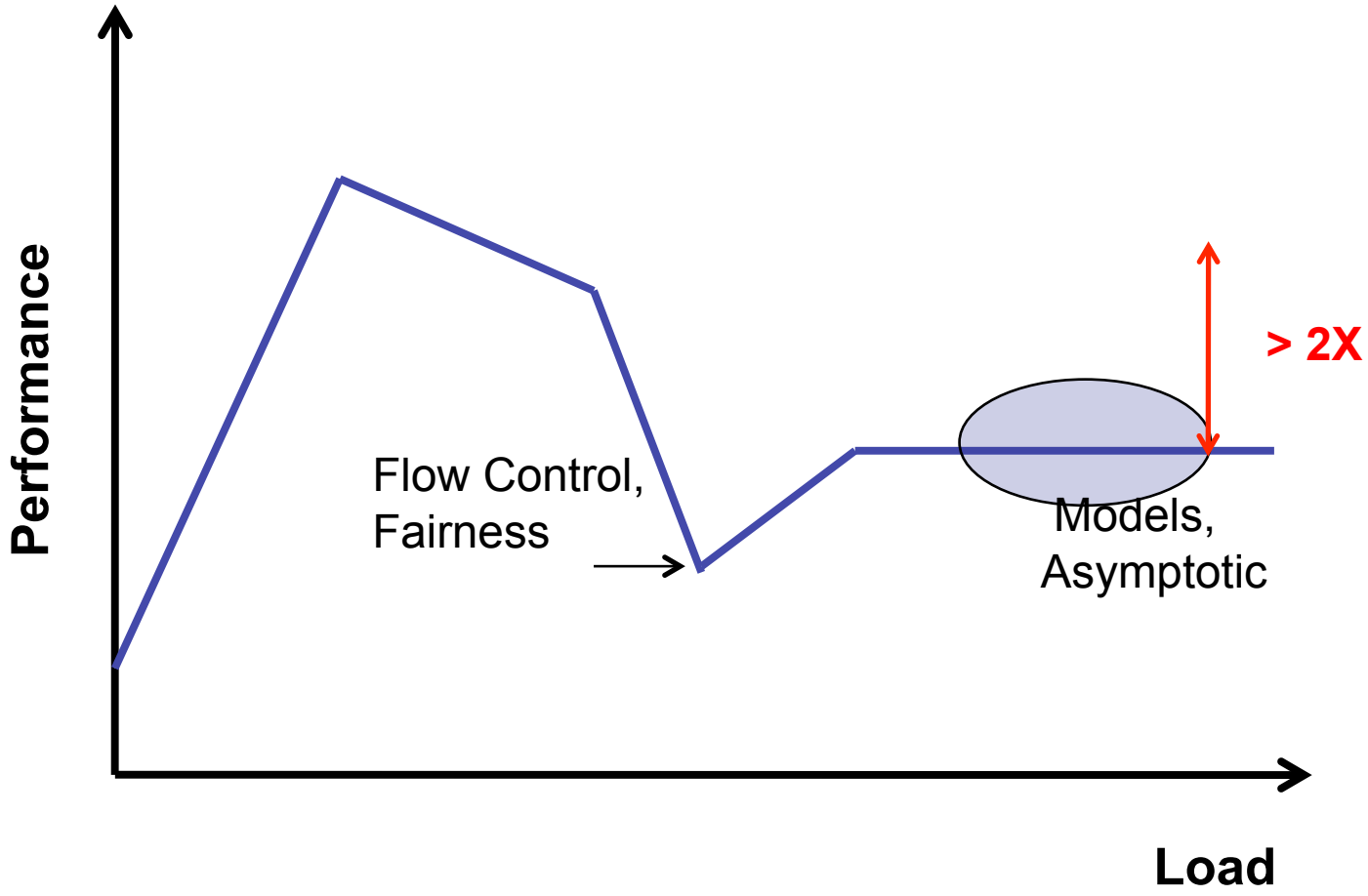
- **Compile and runtime analysis of UPC loop nests**
  - Compiler analyses loop nest and generates templates/stubs annotated with information about behavior (memory region access – LMAD)
  - Runtime analysis decides structure of the transformed code and communication optimizations
  - Communication optimizations are performed using performance models

Performance Portable Optimizations for Loops Containing Communication Operations. Iancu, Chen, Yelick. ICS 2008

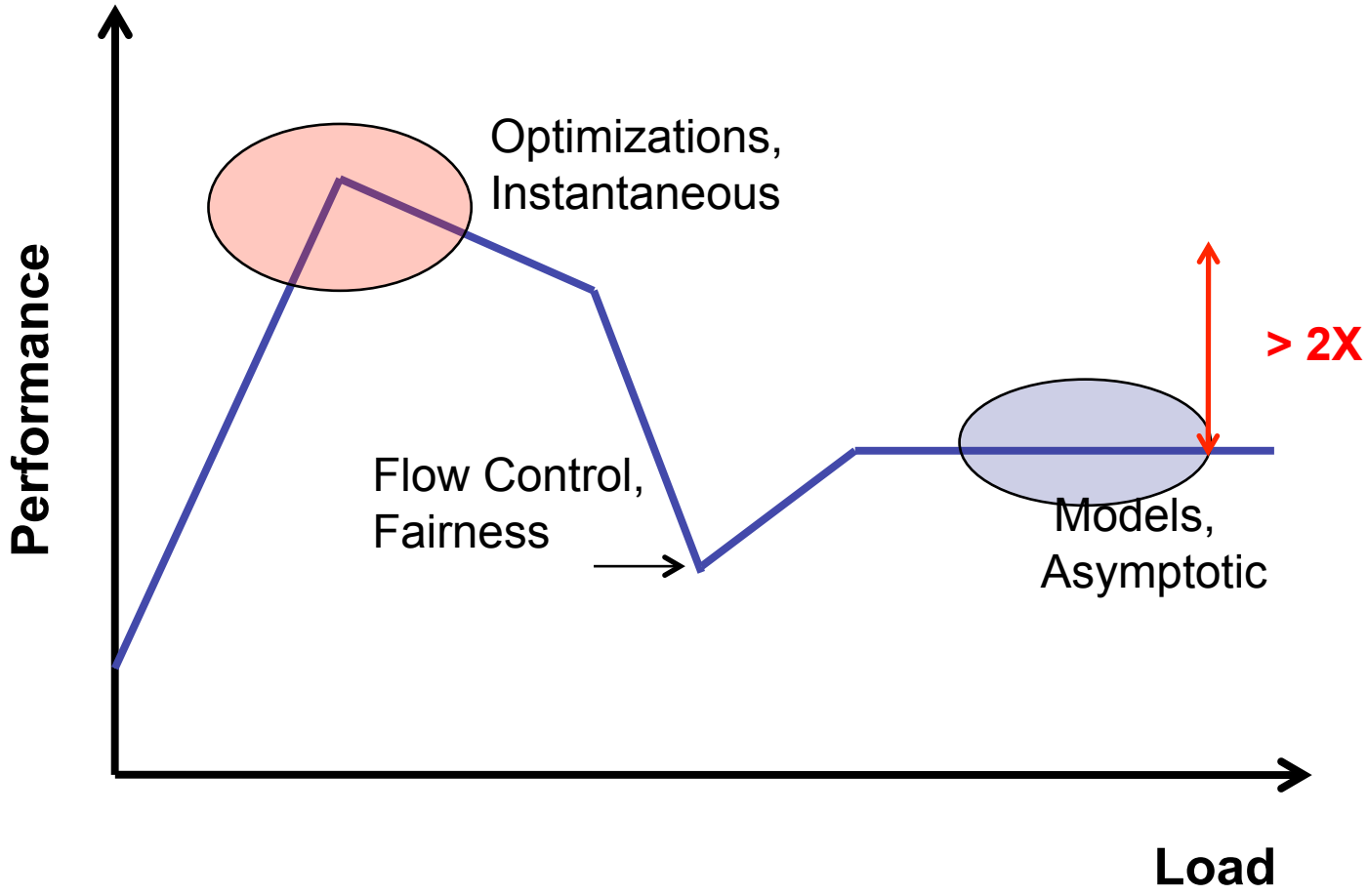
Challenge: Program Representation

- **Optimizer friendly program representation**
  - Experience describing memory regions and flow control
  - What about unstructured parallelism? (DAGs)
  - What about resource requirements/usage?

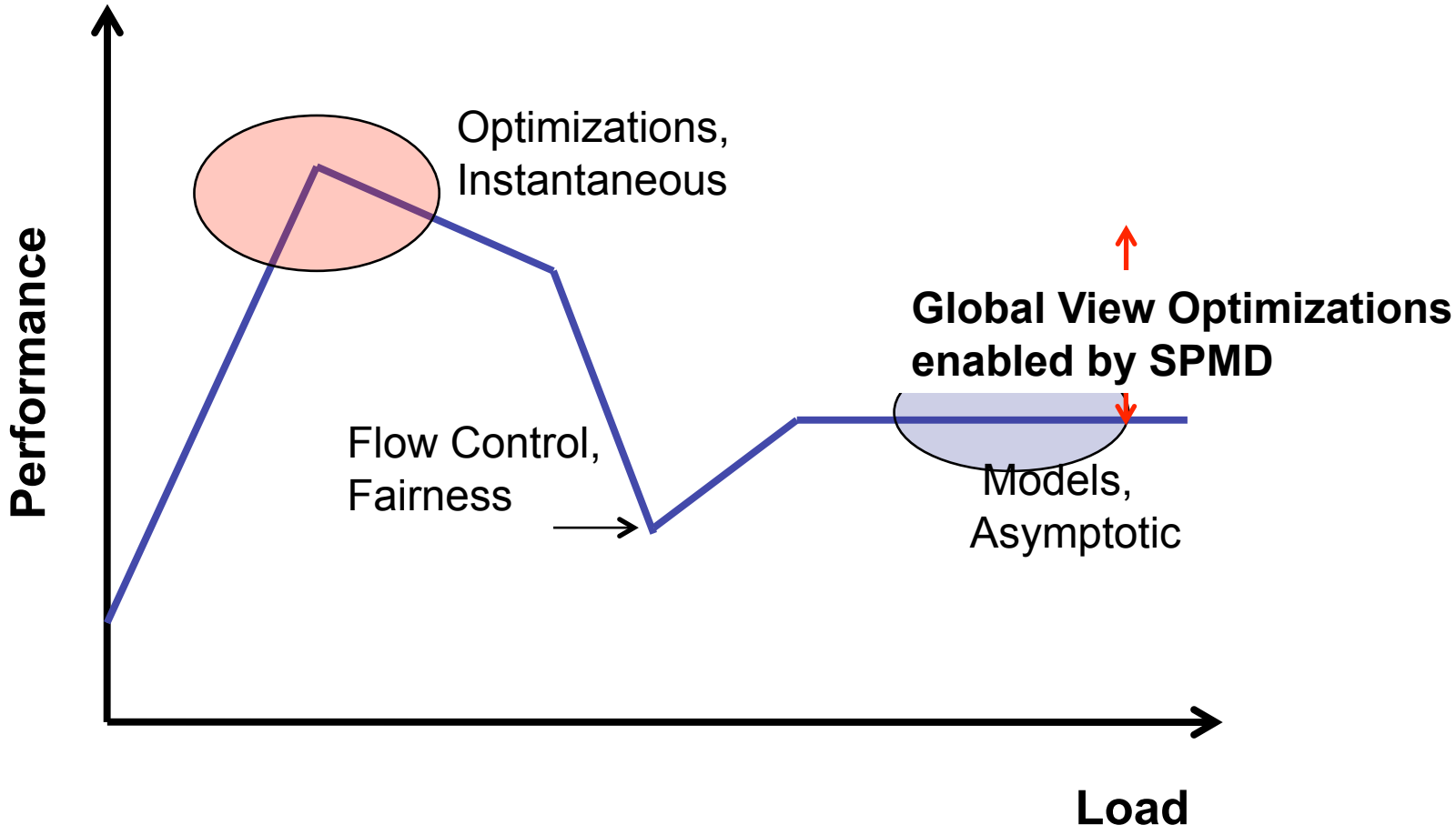
Challenge: Optimization Strategy



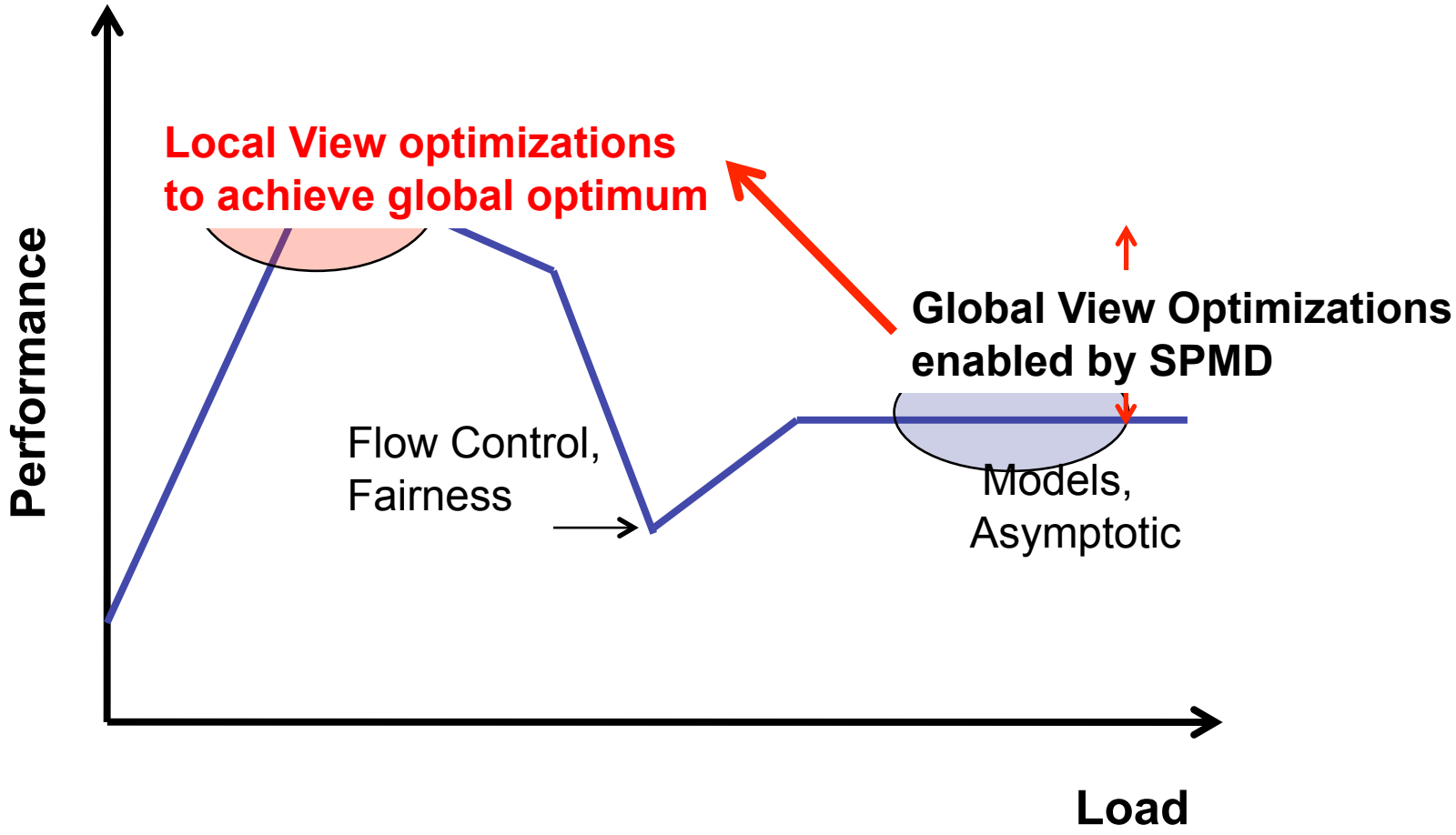
Challenge: Optimization Strategy



Challenge: Optimization Strategy



Challenge: Optimization Strategy



Using Network Performance Models

- **Most approaches measure asymptotic values, optimizations need instantaneous values**
- **Existing "time accurate" performance models do not account well for system scale OR wide SMP nodes**
- **Qualitative models: which is faster, not how fast!**
  (PPoPP'07, ICS'08, PACT'08)
  - Not time accurate, understand errors and model robustness, allow for imprecision/noise, preserve order, be pessimistic

Building an Optimizer

- **Build catalogue of representative scenarios/codes (e.g. all-to-all)**
  - Spatial-temporal exploration of network performance
    - Short and large time scales – account for variability and system noise
    - Small and large system scales – SMP node, full system
  - Understand worst case behavior – **BUILD REPELLER**
- **Develop optimized implementations of representatives using local knowledge**
- **Develop program analysis, representations and dynamic classification schemes to map programs to representatives (pattern matching)**
- **Develop statistical/empirical approaches for optimizations using local knowledge**
  - E.g. combinations of small and large messages

My DEGAS ToDo List

- **Efficient execution at Exascale requires a network centric approach**
  - Message throughput
  - Dynamic communication optimizations
- **Providing message throughout requires**
  - Better OS support
  - Dynamic end-point concurrency control
- **Dynamic optimizations require** (Years 2-3)
  - Better program representations that capture resource usage
  - Different performance models
  - Optimization algorithms using local knowledge

Thank You!

Backup…

Hardware Trends

- **NIC on die (memory controller)**
  - Faster injection is bad for throughput
- **Acceleration** (IBM BG/Q progress thread, Mellanox FCA)
  - NIC still has to match core level of parallelism
- **Tapered networks (asymmetric, not fully connected)**
  - Smaller bisection means lower throughput
- **Where's the flow control?**
- **And NO, hybrid programming won't solve the problem!**
  - Hardware can be really fast, still have to implement high level semantics