Federated Architectures
Lecture 13

Course Summary

Today

How to build available, scalable applications
- Two basic methods

Architectures and contexts
- For building FT, scalable apps.

Future
- Where will the universe be in 10 years?
- What architectures and methods will survive? Why?

Building Applications

How to build a fault tolerant, scalable applications?

- vs. traditional scientific "grand challenge" applications?
- The models used will drive the lower layers

Method 1 :SMPs + DBs + mirroring

- Classic apps, e-bay, e-trade ...

Method 2: Build from scratch on a cluster

- Inktomi, Porcupine, NOW-sort

Build from scratch on an SMP?

Recall 3 systems issues

Scalability

- Speedup (various definitions)
  - time constrained
  - resource constrained
  -
- Short term (minutes, hours)
- Long term (years)

Availability

- Resistance to all faults

Manageability

- Human effort needed to keep system running

Why these apps are hard

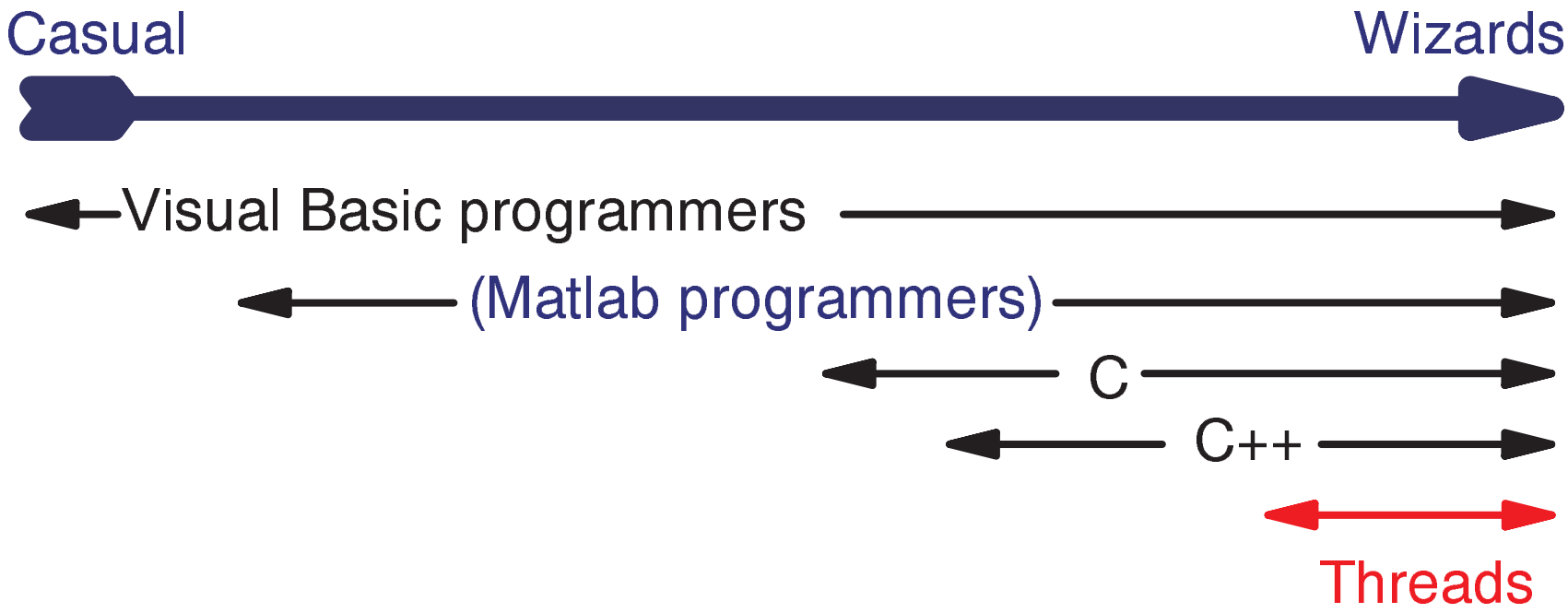
Rely on parallelism
- for performance and availability

Someone must partition data and control for all classes of machines
- Relational model easier, but still requires care (E.g. Oracle and Teradata programmers)

Fundamentally harder than normal programming (essence vs accident)

Add reasoning about faults in the mix
- What happens when component X fails?

Ousterhout's Taxonomy



- Too hard to use
- Even for experts, development is painful

Why parallelism is hard

Adds new problems irrespective of parallel programming model

- Data partitioning
- Scheduling control
  - Event handling
  - Synchronization
- Deadlock
- Race conditions
- Performance
- Debugging a nightmare

Concurrency adds complexity

When is it worth the cost to move to a parallel/distributed programming model?

Availability demands

- 99.9999% uptime (< 1 min per year)

Scalability (Performance) demands

- Constant response time
- Scalable throughput
- Volume of data (Multiple Tera ....)

Manageability demands? (reverse?)

Design Points

cc-NUMA
- SMP OS
- cell based OS

Cluster
- with μ-kernel
- whole kernel

Federated Architecture
- embedded OS
- μ-kernel based OS

cc-NUMA + SMP OS

Tradition which new systems must measure up to

Scalability

- Traditional scaling problems "solved"
- Scaling over time still a problem

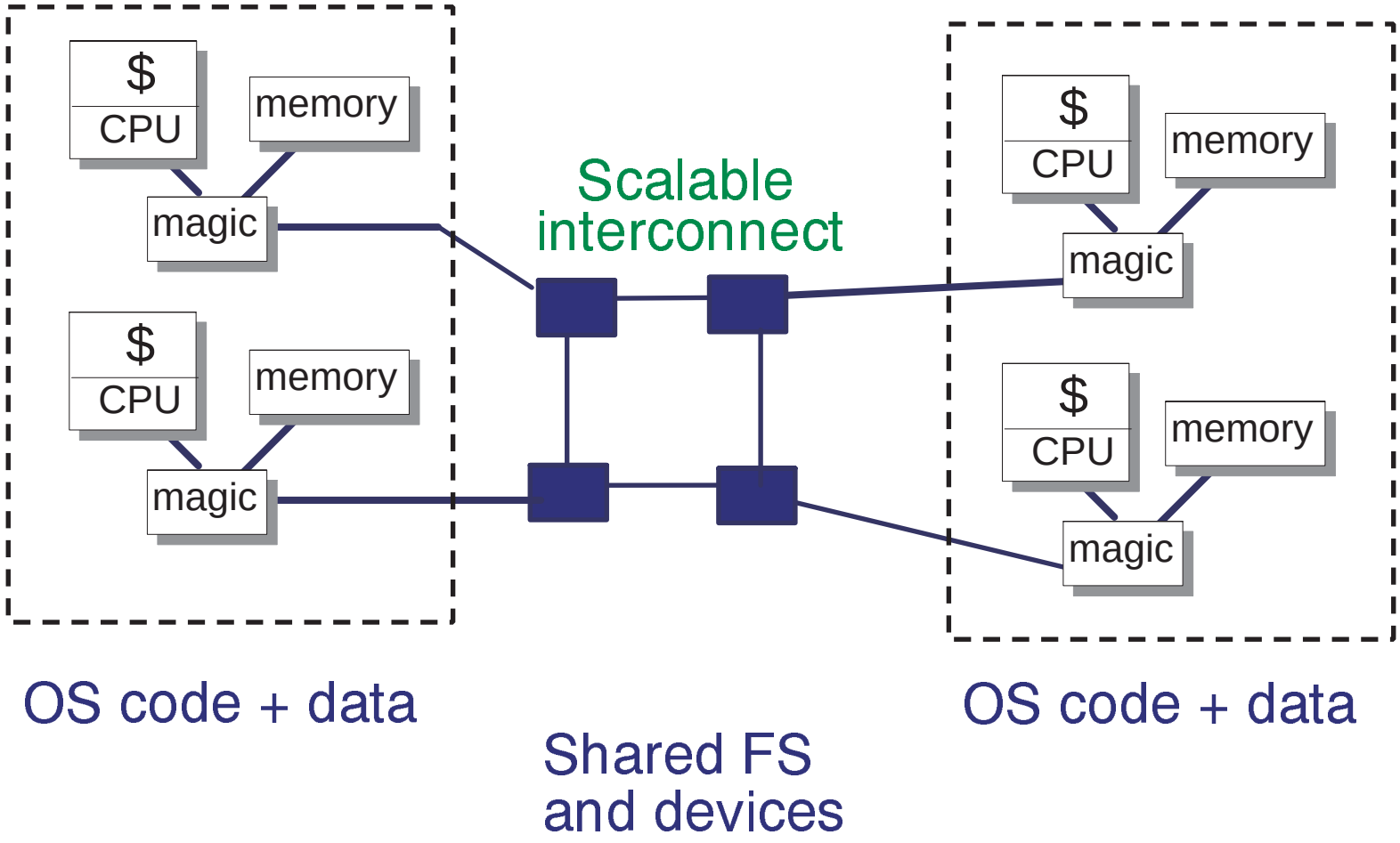
Availability

- Still a big problem

Manageability

- Reuse of OS a huge win
  - Devices
  - Job control
  - Security and authentication

ccNUMA w. Cells



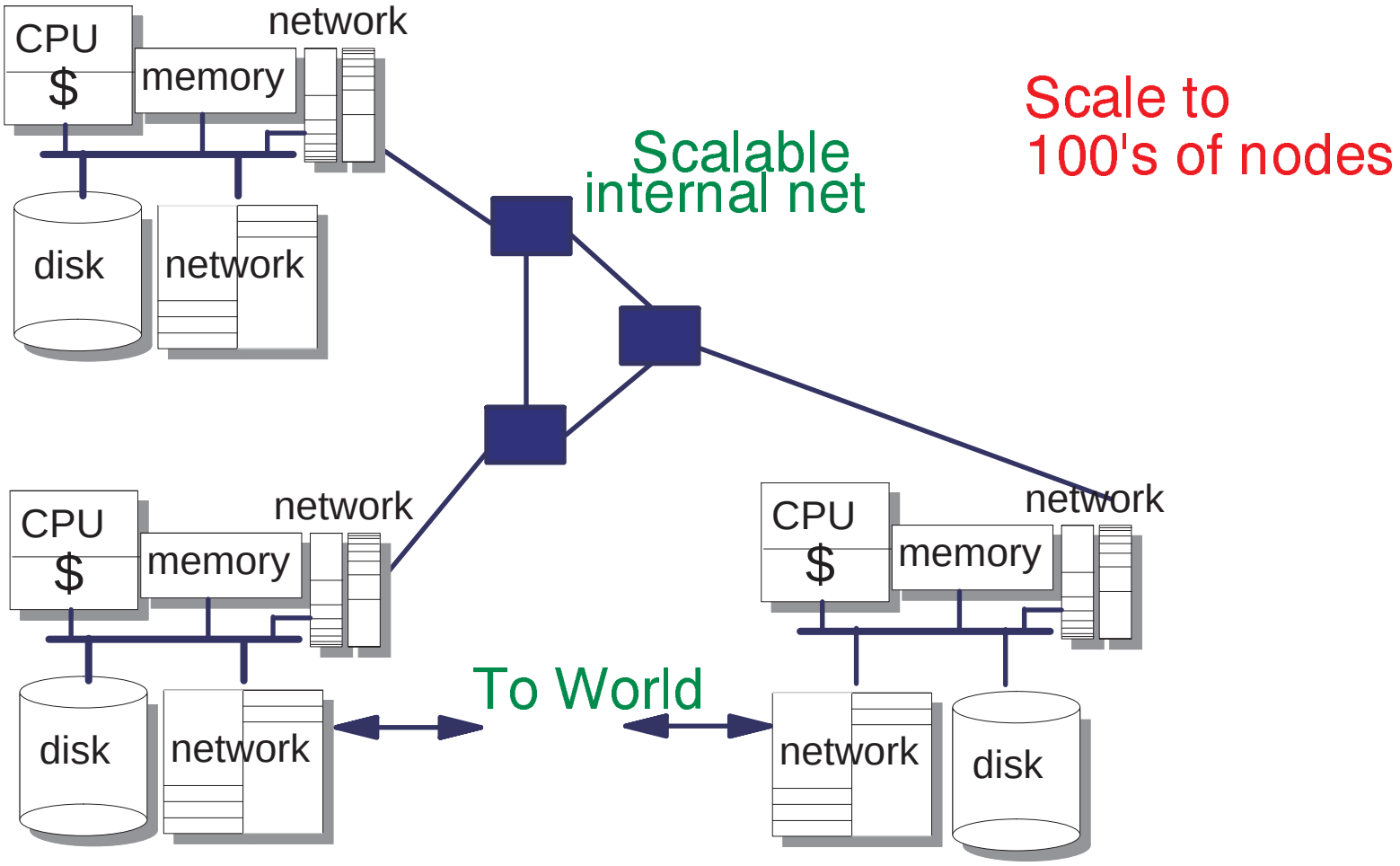
cc-NUMA + Cells

Scalability
- Pro's of SMPs

Availability of OS much higher
- But application availability no better

Manageability of system slightly worse than SMP?
- Cells a new level of abstraction to maintain
- Worth the trade-off?

Cluster


CPU
$
memory
network
disk
network
Scalable
internal net
Scale to
100's of nodes
CPU
$
memory
network
disk
network
CPU
$
memory
network
network
disk
To World

Cluster + μ-kernel

Most scalable architecture

Current availability is poor

Manageability

- Key are the OS servers
  - scalable FS, VM, Communications

Single system image huge win

- μ-kernel on same track as normal kernel?
- Not re-using engineering effort

Cluster + Normal Kernel

- "Linux is a μ-kernel"
  - Agree with this statement?
  - Can we strip a normal kernel to the bare essentials and get "the best of both"?
- Still have to partition the application
- How much does a scalable OS actually help the application developer?
  - if they have to partition anyway
- Lack of global services a nightmare
  - E.g. Security holes

A Cluster approach?

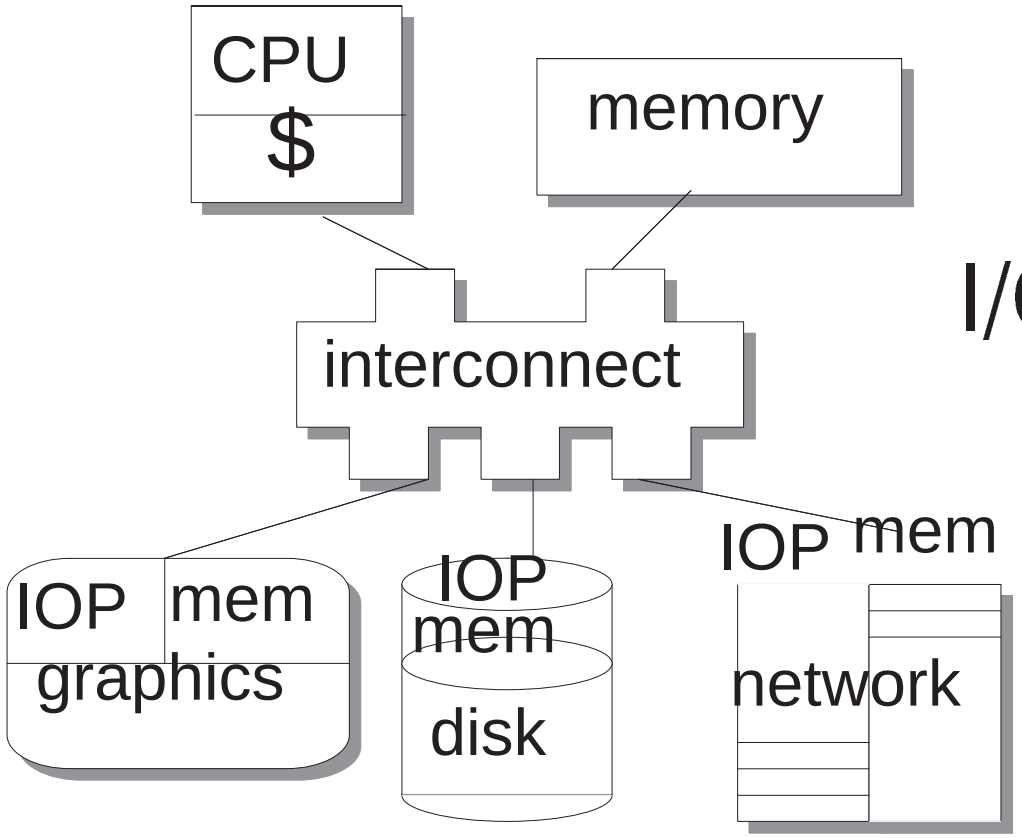
Still require the global services like(μ-kernel)

- name service
- job control
- file system
- storage

OS solution: build down instead of up

- Use a regular kernel but stripped of what you don't want

Federated Architecture



I/O processors

Federated + embedded

I/O processors key distinction

SMPs will move in this direction (slowly)

- depends on your definition of "processor"
- Vendors will not open up much outside programming

Limited Scalability

Needs standard off-board execution environment

- Otherwise, too hard to program
- I20 was a start
- Will it become mainstream?

Federated + μ-kernel

- Each node runs the same m-kernel
- Servers run on main CPU's
- I/O processors run on devices
  - Use SMP nodes or not?
- Same problems as cluster case

Scalable OS abstractions

Applications:

- Inktomi
- Porcupine
- DB Kernels (Sort, Join, X-pose)

File System

- xFS, Frangipani, GFS, Teradata DB

Storage System

- Petal, swarm

How would the storage abstractions help/hinder application development?

Availability

Achilles s heel of SMPs
- How much does HA buy you?
  - A lot, but is it enough?
- Cell-based approaches?

Potential strength of clusters
- But current systems fail badly

Potential of Fed. Architecture
- But little in the way of models or software

A universe of bad things

What is are the set of perturbations/faults designers should care about?

- Hardware
- Software
- Environment
- Operations
- Performance

When was the last time you saw an application with these factors included in the design?

Future

Next large scale computing platform?

- Incremental
  - SMP+ Cells?
- Evolution
  - Cluster/Federated with μ-kernel?
- Revolution
  - Some new architecture and OS?

Factors

Technical

- Performance
- Scalability
- Manageability
- Engineering effort (lag = performance loss)

Nontechnical

- Historic
  - 10X rule: new tech. must be 10x better
- Business
  - no one got fired for buying {IBM, Microsoft,Sun}

El Dorado System

Scalability of a cluster

- 100's of processors
- TB of memory
- 10^3 - 10^4 disks
- Scalability through technologies

Availability of a Tandem

- Resistance to HW and SW faults
- changes in technology
- Operations and environmental changes

Maintainability of an SMP

- Single system image
- Single FS, Port and Device namespaces

Projects

- Due Dec 17th
- Expect a reasonable write-up
  - Some working system + experimental data
- Let me know if you're having problems