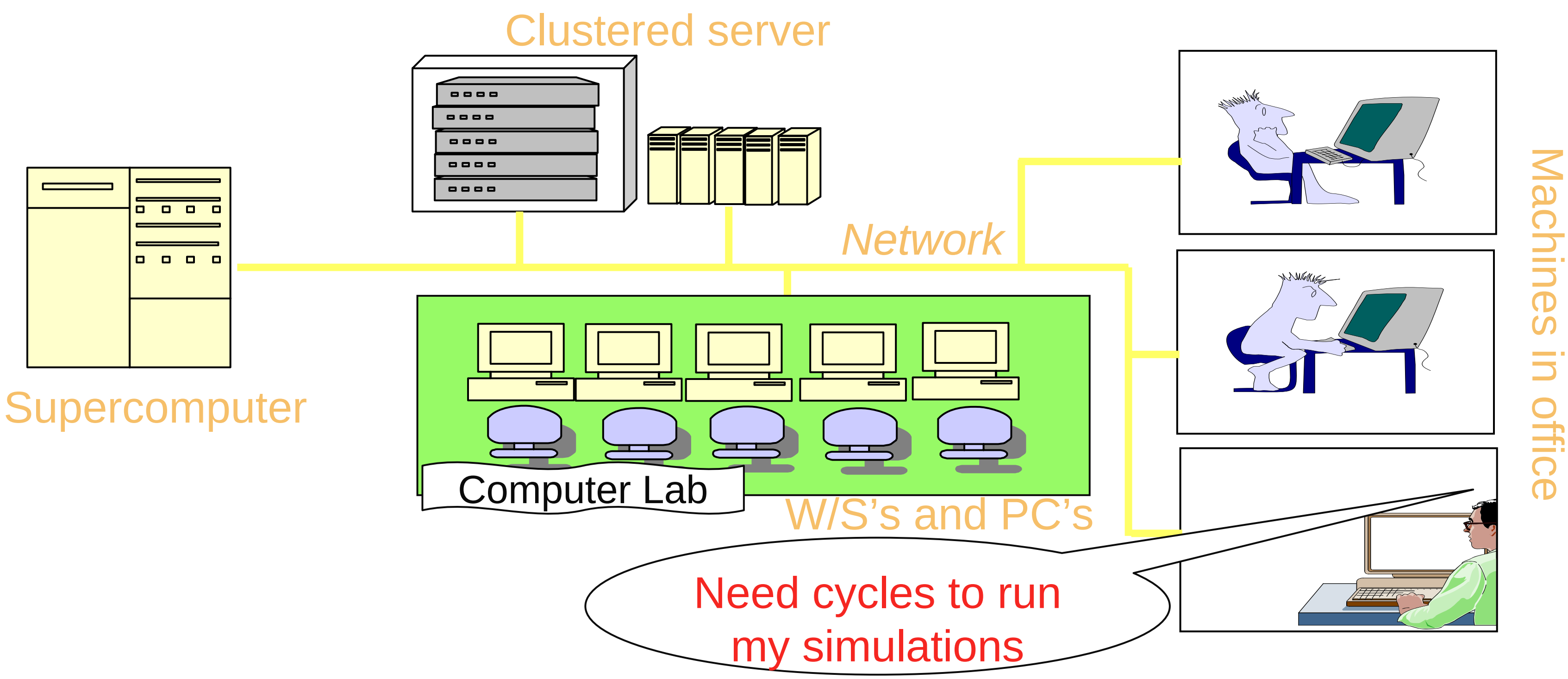
Computing Environment

- Cost Effective High Performance Computing
  - Dedicated servers are expensive
  - Non-dedicated machines are useful
    - high processing power(~1GHz), fast network (100Mbps+)
    - Long idle time(~50%), low resource usage



OS Support For Parallel Computing

- Many applications need raw compute power
  - Computer H/W and S/W Simulations
  - Scientific/Engineering Computation
  - Data Mining, Optimization problems
- Goal
  - Exploit computation cycles on idle workstations
- Projects
  - Condor
  - Linger-Longer

Issues

- Scheduling
  - What jobs to run on which machines?
  - When to start / stop using idle machines?
- Transparency
  - Can applications execute as if on home machine?
- Checkpoints
  - Can work be saved if job is interrupted?

What Is Condor?

- Condor
  - Exploits computation cycles in collections of
    - workstations
    - dedicated clusters
  - Manages both
    - resources (machines)
    - resource requests (jobs)
  - Has several mechanisms
    - ClassAd Matchmaking
    - Process checkpoint/ restart / migration
    - Remote System Calls
    - Grid Awareness
  - Scalable to thousands of jobs / machines

Condor – Dedicated Resources

- Dedicated Resources
  - Compute Clusters
- Manage
  - Node monitoring, scheduling
  - Job launch, monitor & cleanup

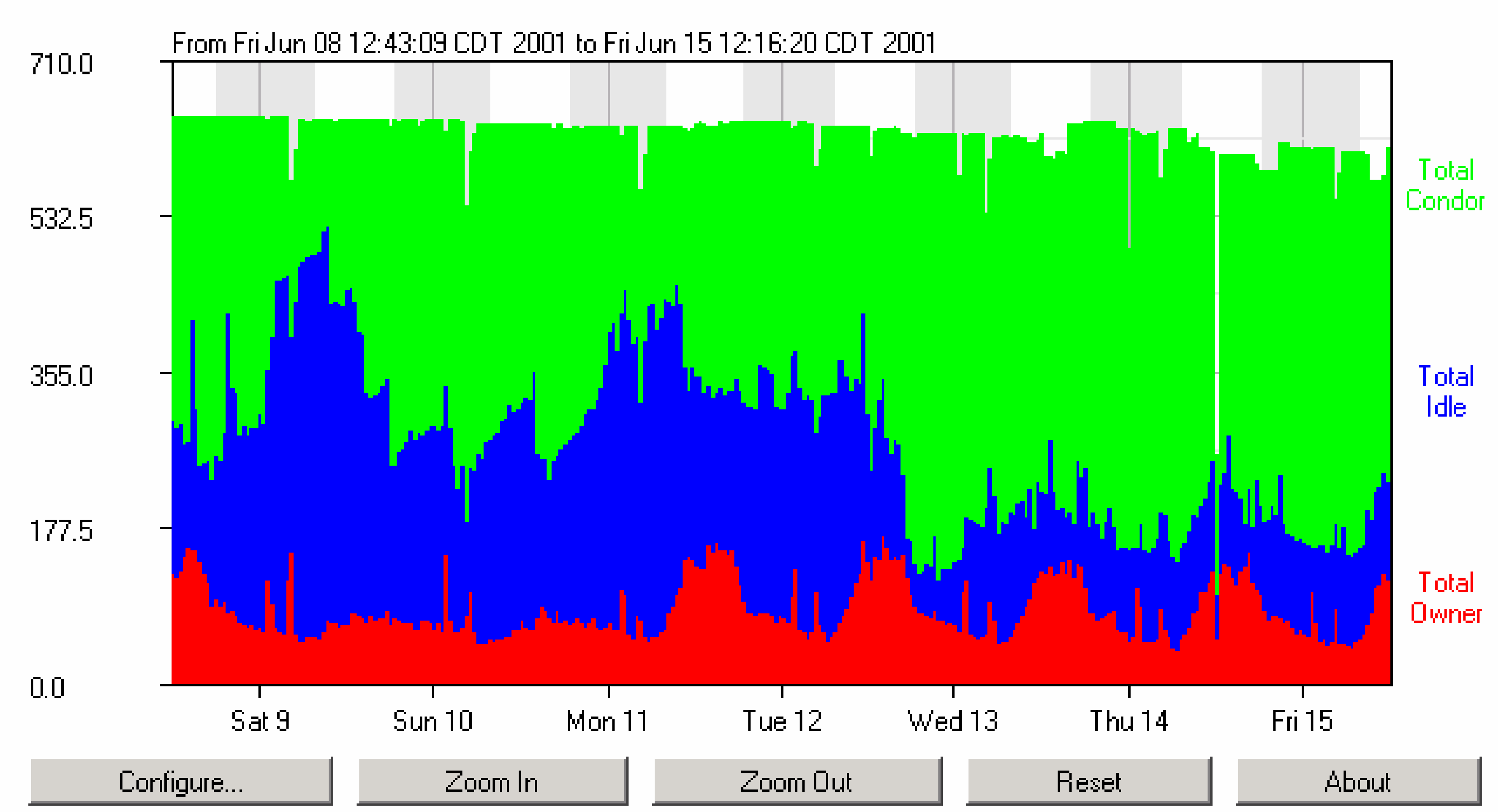
Condor – Non-dedicated Resources

- Examples
  - Desktop workstations in offices
  - Workstations in student labs
- Often idle
  - Approx 70% of the time!
- Condor policy
  - Use workstation if idle
  - Interrupt and move job if user activity detected

Mechanisms in Condor

- Transparent Process Checkpoint / Restart
- Transparent Process Migration
- Transparent Redirection of I/O
  - Condor's Remote System Calls

CondorView Usage Graph



What is ClassAd Matchmaking?

- Condor uses ClassAd Matchmaking to make sure that work gets done within the constraints of both users and owners.
- Users (jobs) have constraints:
  - "I need an Alpha with 256 MB RAM"
- Owners (machines) have constraints:
  - "Only run jobs when I am away from my desk and never run jobs owned by Bob."
- Semi-structured data --- no fixed schema

Some Challenges

- Condor does whatever it takes to run your jobs, even if some machines…
  - Crash (or are disconnected)
  - Run out of disk space
  - Don’t have your software installed
  - Are frequently needed by others
  - Are far away & managed by someone else

Condor's Standard Universe

- Condor can support various combinations of features/environments
  - In different "Universes"

- Different Universes provide different functionality
  - Vanilla
    - Run any Serial Job
  - Scheduler
    - Plug in a meta-scheduler
  - Standard
    - Support for transparent process checkpoint and restart

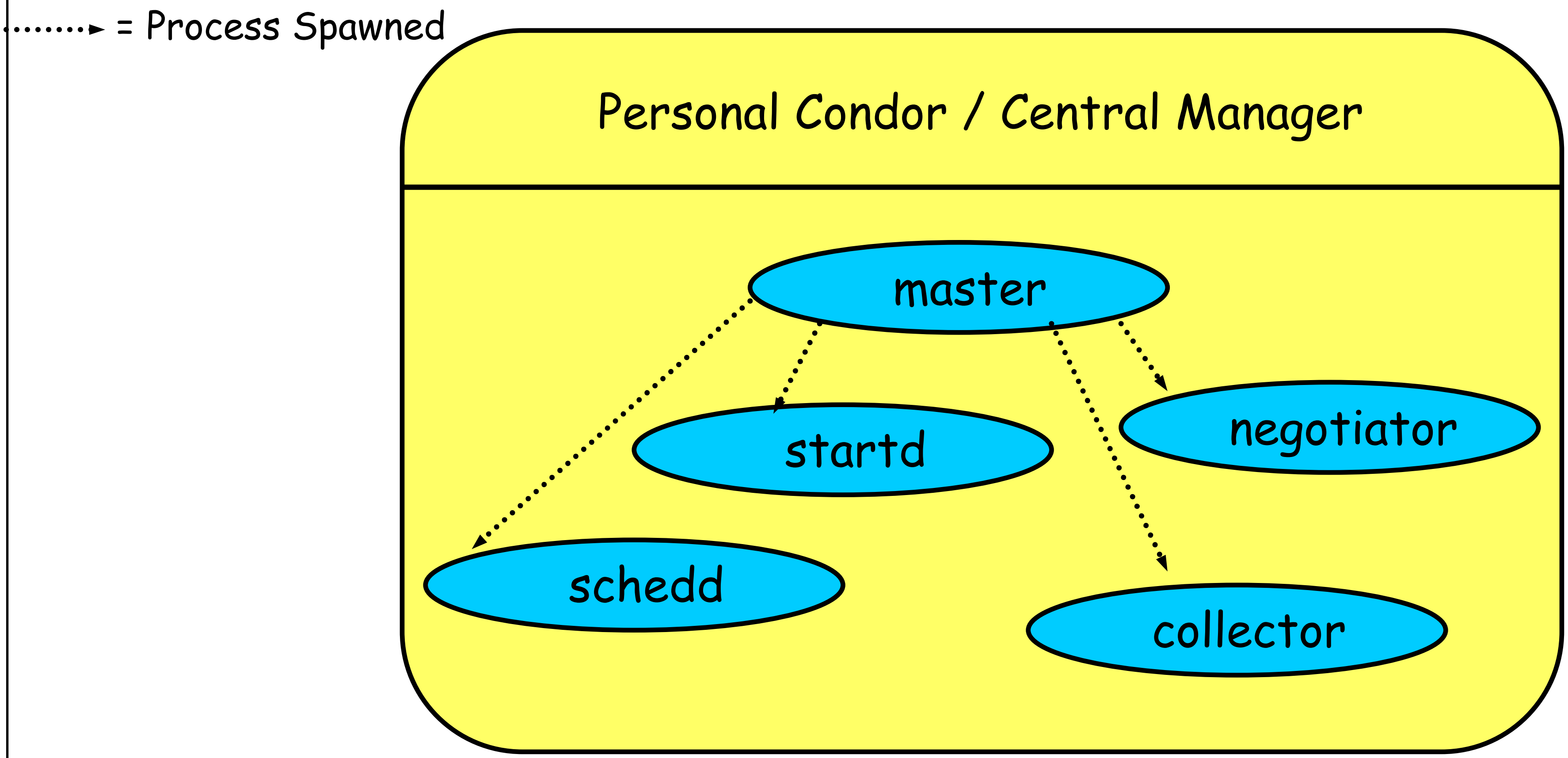
Process Checkpointing

- Condor's Process Checkpointing mechanism saves all the state of a process into a checkpoint file
  - Memory, CPU, I/O, etc.
- The process can then be restarted
  - From right where it left off
- Typically no changes to your job's source code needed
  - However, your job must be relinked with Condor's Standard Universe support library

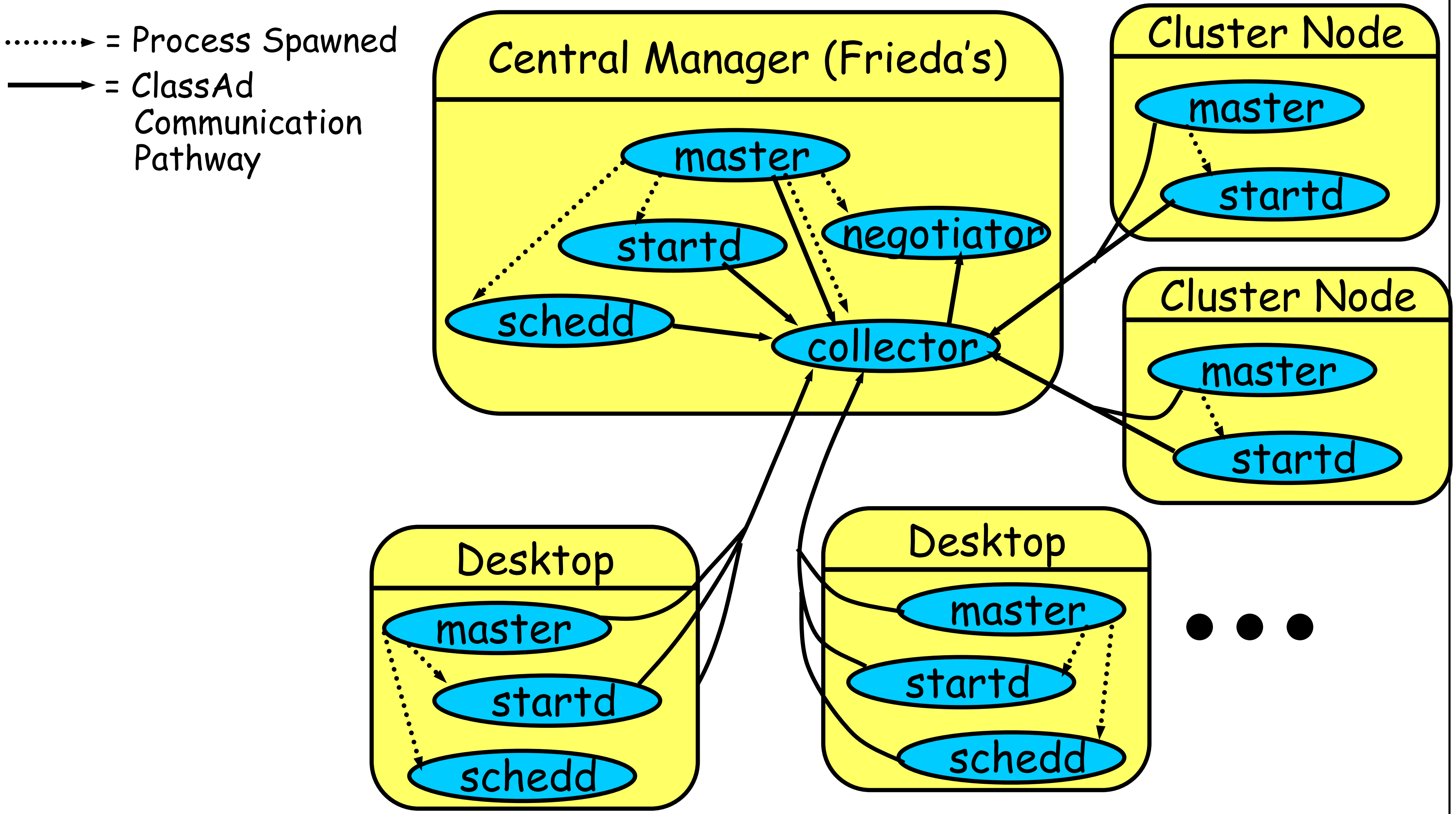
When Will Condor Checkpoint Your Job?

- Periodically, if desired
  - For fault tolerance
- To free the machine to do a higher priority task (higher priority job, or a job from a user with higher priority)
  - Preemptive-resume scheduling
- When you explicitly run
  - condor_checkpoint
  - condor_vacate
  - condor_off
  - condor_restart

Condor Daemon Layout



Layout of the Condor Pool



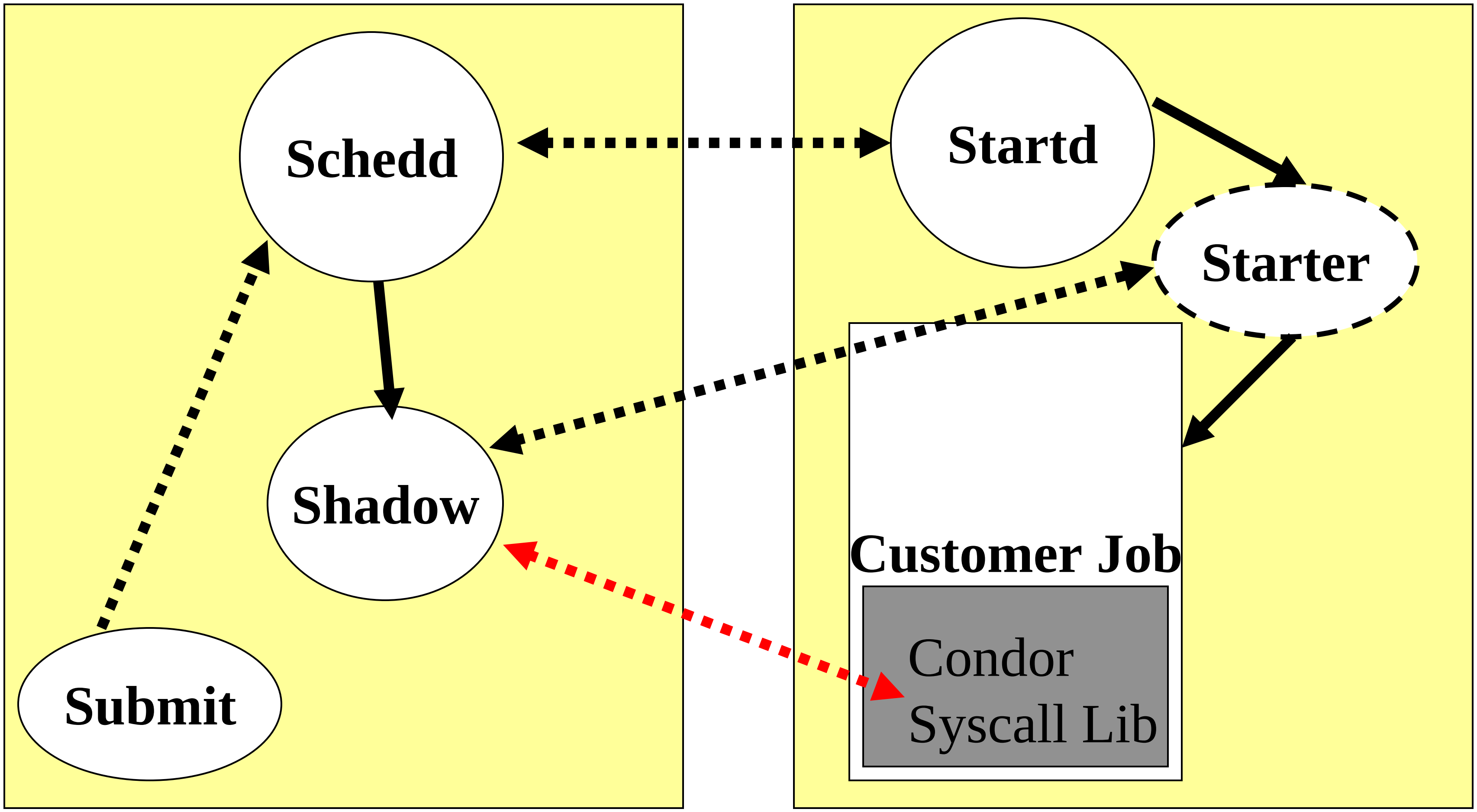
Access to Data in Condor

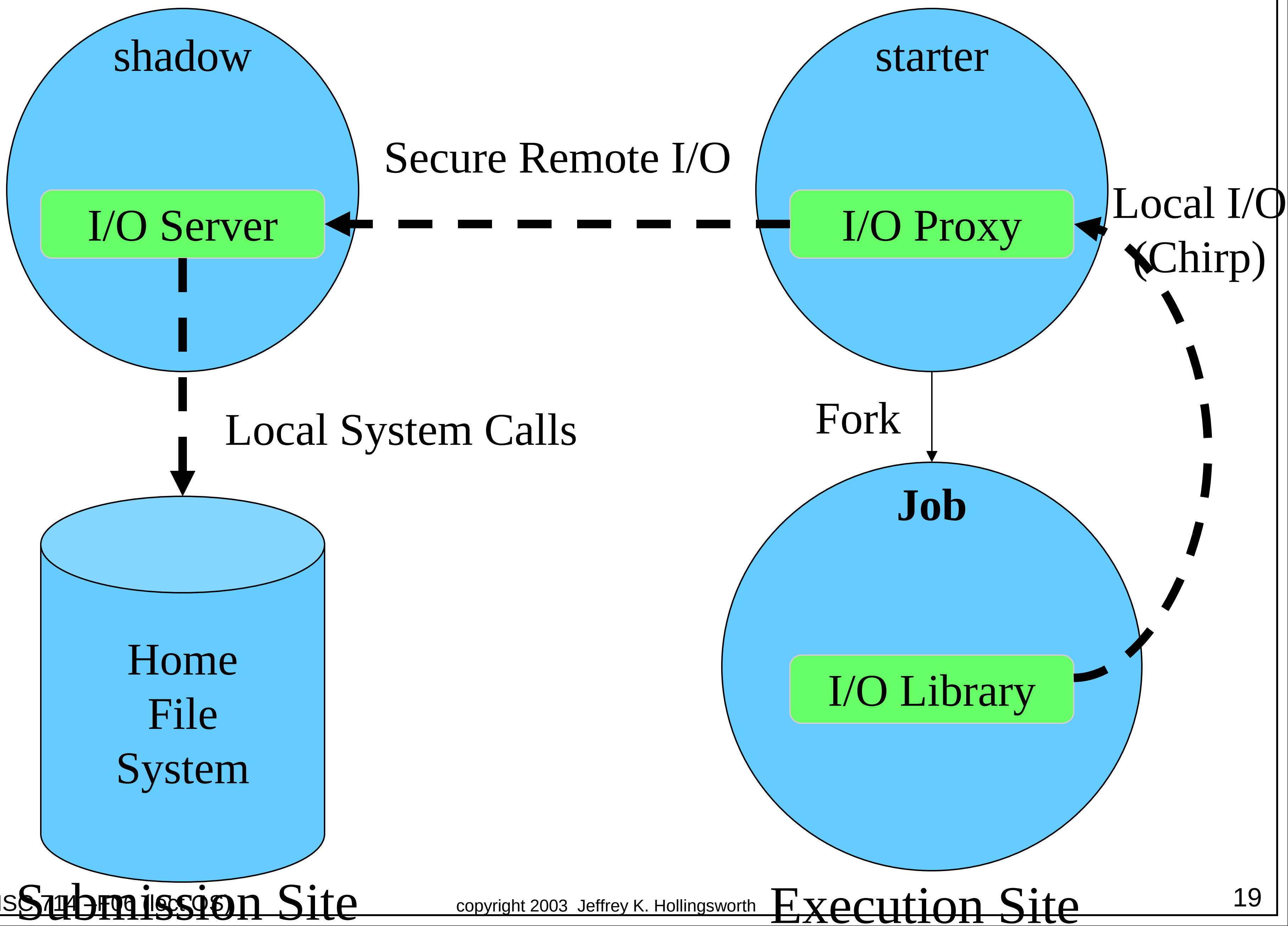
- Use Shared Filesystem if available
- No shared filesystem?
  - *Remote System Calls (in the Standard Universe)*
  - Condor File Transfer Service
    - Can automatically send back changed files
    - Atomic transfer of multiple files
  - Remote I/O Proxy Socket

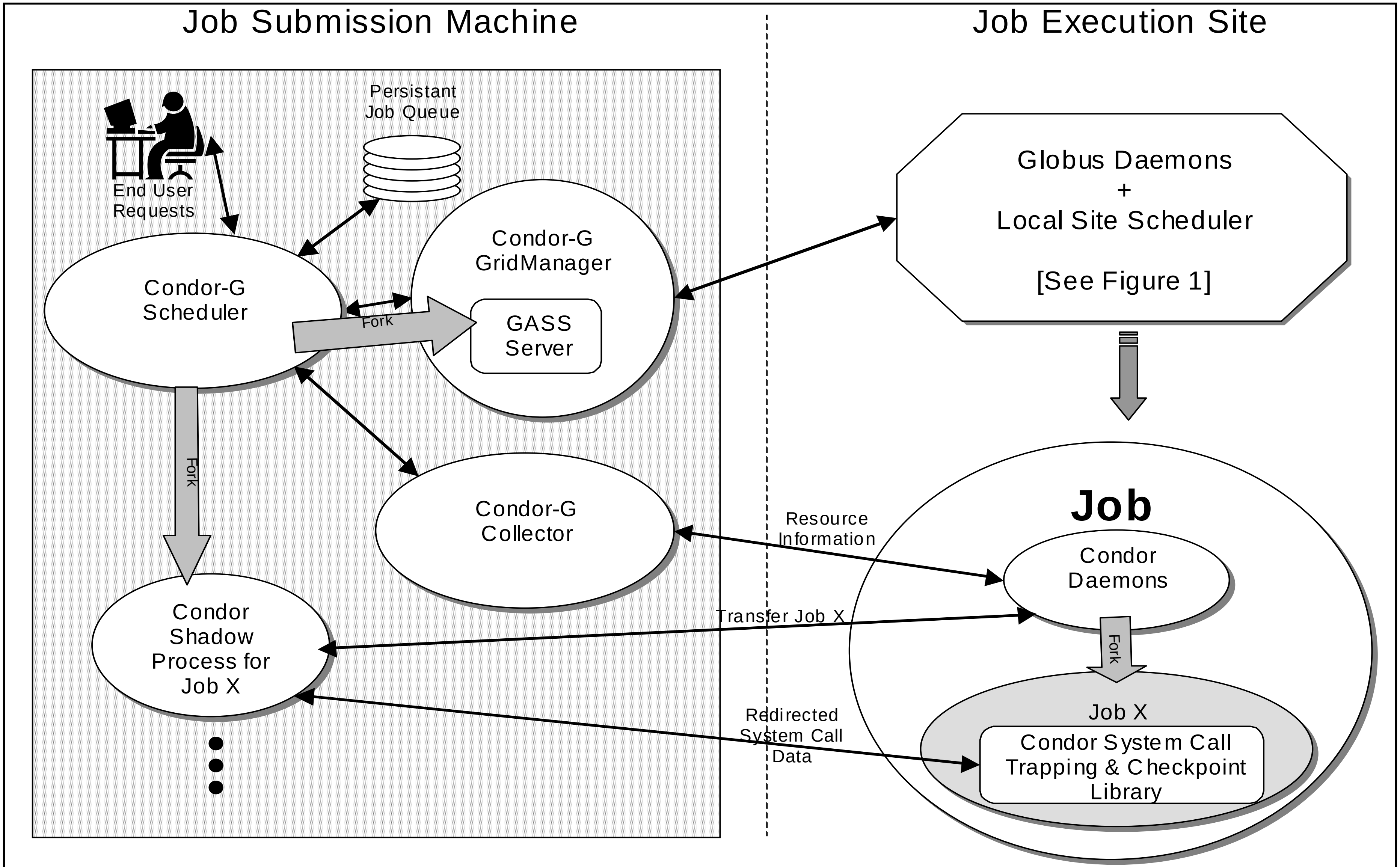
Standard Universe Remote System Calls

- I/O System calls trapped
  - Sent back to submit machine
- Allows Transparent Migration Across Domains
  - Checkpoint on machine A, restart on B
- No Source Code changes required
- Language Independent
- Opportunities
  - For Application Steering
    - Condor tells customer process “how” to open files
  - For compression on the fly

Job Startup


Schedd
Startd
Starter
Shadow
Customer Job
Condor
Syscall Lib
Submit


shadow
I/O Server
Secure Remote I/O
starter
I/O Proxy
Local I/O
(Chirp)
Local System Calls
Fork
Job
I/O Library
Home
File
System
Submission Site
Execution Site



Job Submission Machine
Job Execution Site
Persistant Job Queue
End User Requests
Condor-G Scheduler
Condor-G GridManager
GASS Server
Fork
Condor-G Collector
Condor Shadow Process for Job X
Globus Daemons + Local Site Scheduler [See Figure 1]
Resource Information
Transfer Job X
Redirected System Call Data
Job
Condor Daemons
Job X
Condor System Call Trapping & Checkpoint Library

Exploiting Idle Cycles in Networks of Workstations

Kyung Dong Ryu

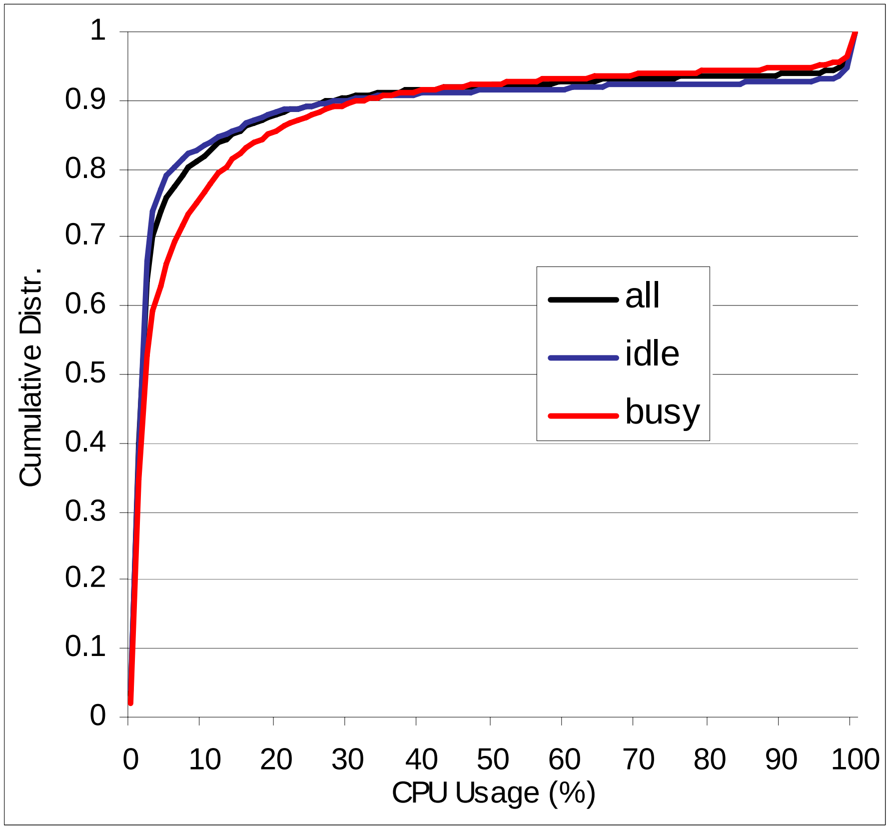
UNIVERSITY OF MARYLAND

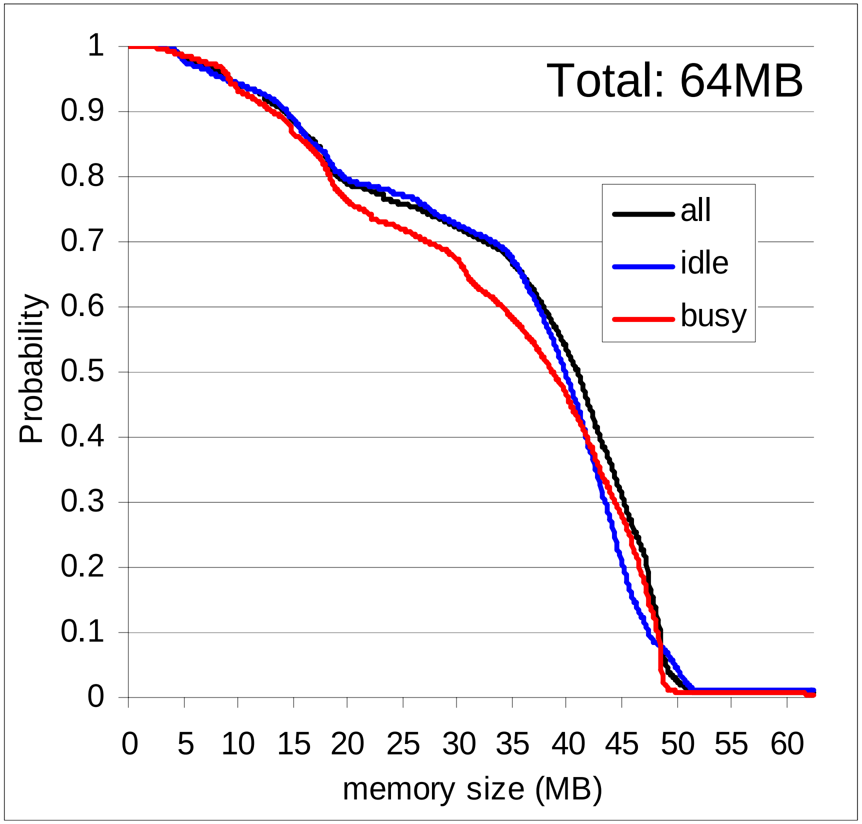
High Performance Computing in NOW

- Many systems support harvesting idle machines
  - Traditional Approach : Coarse-Grained Cycle Stealing
    - while owner is away: send guest job and run
    - when owner returns: stop, then
      - migrate guest job: Condor, NOW system
      - suspend or kill guest job: Butler, LSF, DQS system
- But…

Additional CPU Time and Memory is Available

- When a user is active
  - CPU usage is < 10%, 75% of time
  - 30 MB memory is available, 70% of time

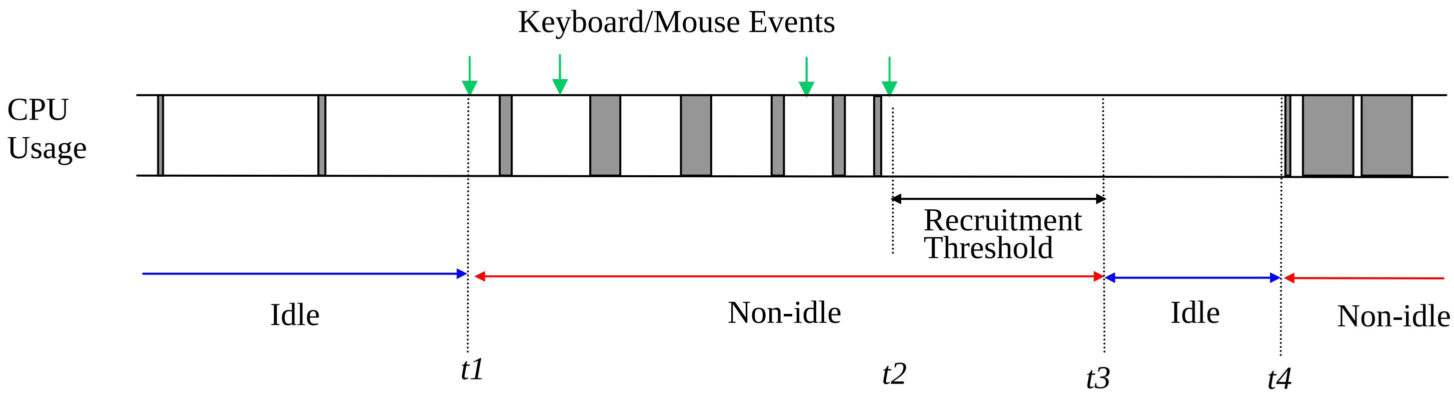




Questions

- Can we exploit fine grained idle resources?
  - For sequential programs and parallel programs
  - Improve throughput
- How to reduce effect on user?
  - Two level CPU scheduling
  - Memory limits
  - Network and I/O throttling

Fine-Grain Idle Cycles



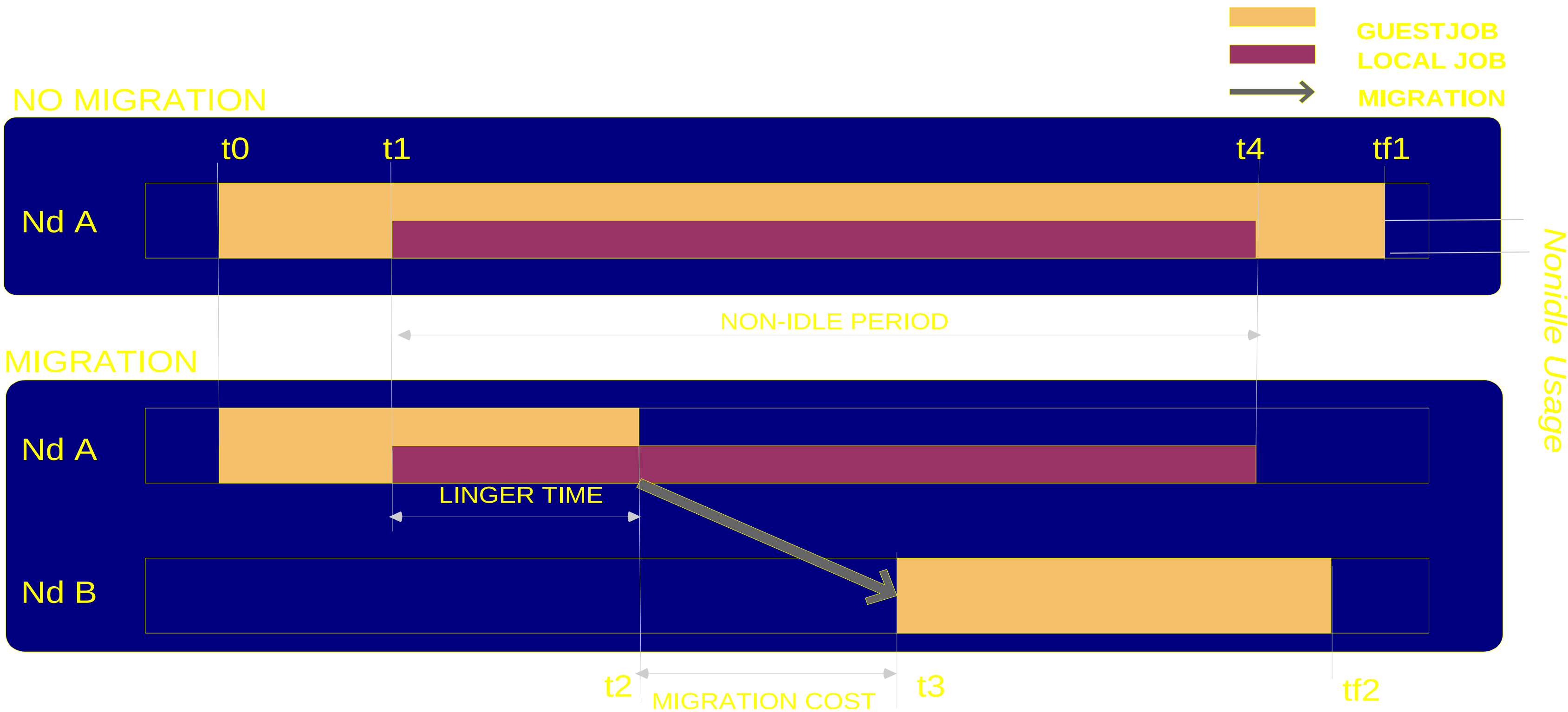
- Coarse-Grain Idle Cycles
  - (t1,t3): Keyboard/mouse events
  - (t4,~): High CPU usage
  - Recruitment threshold
- Fine-Grain Idle Cycles
  - All empty slots
  - Whenever resource(CPU) is not used

Linger Longer: Fine-Grain Cycle Stealing

- Goals:
  - Harvest more available resources
  - Limit impact on local jobs
- Technique: Lower Guest Job's Resource Priority
  - Exploit fine-grained idle intervals even when user is active
    - Starvation-level low CPU priority
    - Dynamically limited memory use
    - Dynamically throttled I/O and network bandwidth
- Adaptive Migration
  - No need to move guest job to avoid local job delay
  - Could move guest job to improve guest performance

Adaptive Migration

- When Migration Benefit overweighs Migration Cost
  - Non-idle_Period ≥ Linger_Time + Migration_Cost / Non-idle_Usage
  - Linger_Time ∝ Migration_Cost / Non-idle_Usage
    - Migration_cost = Suspend_Time(source) + Process_Size/Network_Bandwidth + Resume_Time(dest.)



Need a Suite of Mechanisms

Goal: maximize usage of idle resources

Constraint: limit impact on local jobs

- Policy:
  - Most unused resources should be available
  - Resource should be quickly revoked when local jobs reclaim
- Dynamic Bounding Mechanisms for:
  1. CPU
  2. Memory
  3. I/O Bandwidth
  4. Network Bandwidth

CPU bounding: Is Unix “nice” sufficient ?

- CPU priority is not strict
  - run two empty loop processes (guest: nice 19)

OS	Host	Guest
Solaris (SunOS 5.5)	84%	15%
Linux (2.0.32)	91%	8%
OSF1	99%	0%
AIX (4.2)	60%	40%

- Why ?
  - Anti-Starvation Policy

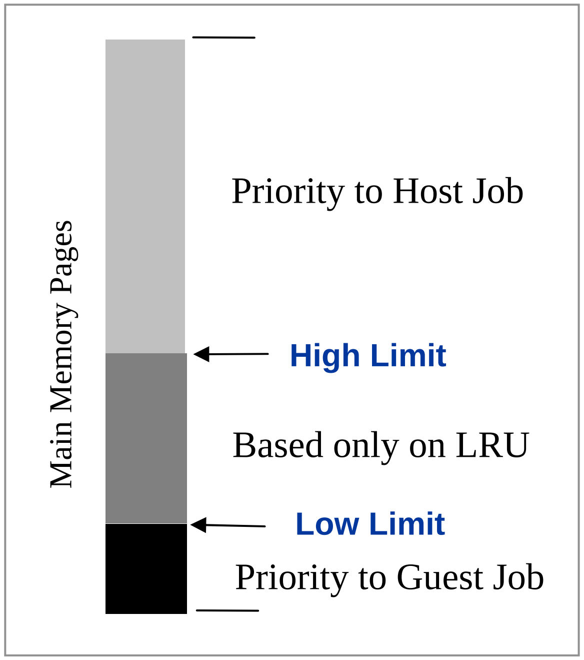
CPU Bounding: Starvation Level Priority

- Original Linux CPU Scheduler
  - One Level : process priority
  - Run-time Scheduling Priority
    - nice value & remaining time quanta
    - $T_i = 20 - nice\_level + 1/2 * T_{i-1}$
  - Low priority process can preempt high priority process

- Extended Linux CPU Scheduler
  - Two Level : 1) process class, 2) priority
  - If runnable host processes exist
    - Schedule a host process as in unmodified scheduler
  - Only when no host process is runnable
    - Schedule a guest process

Memory Bounding: Page Limits

- Extended page replacement algorithm

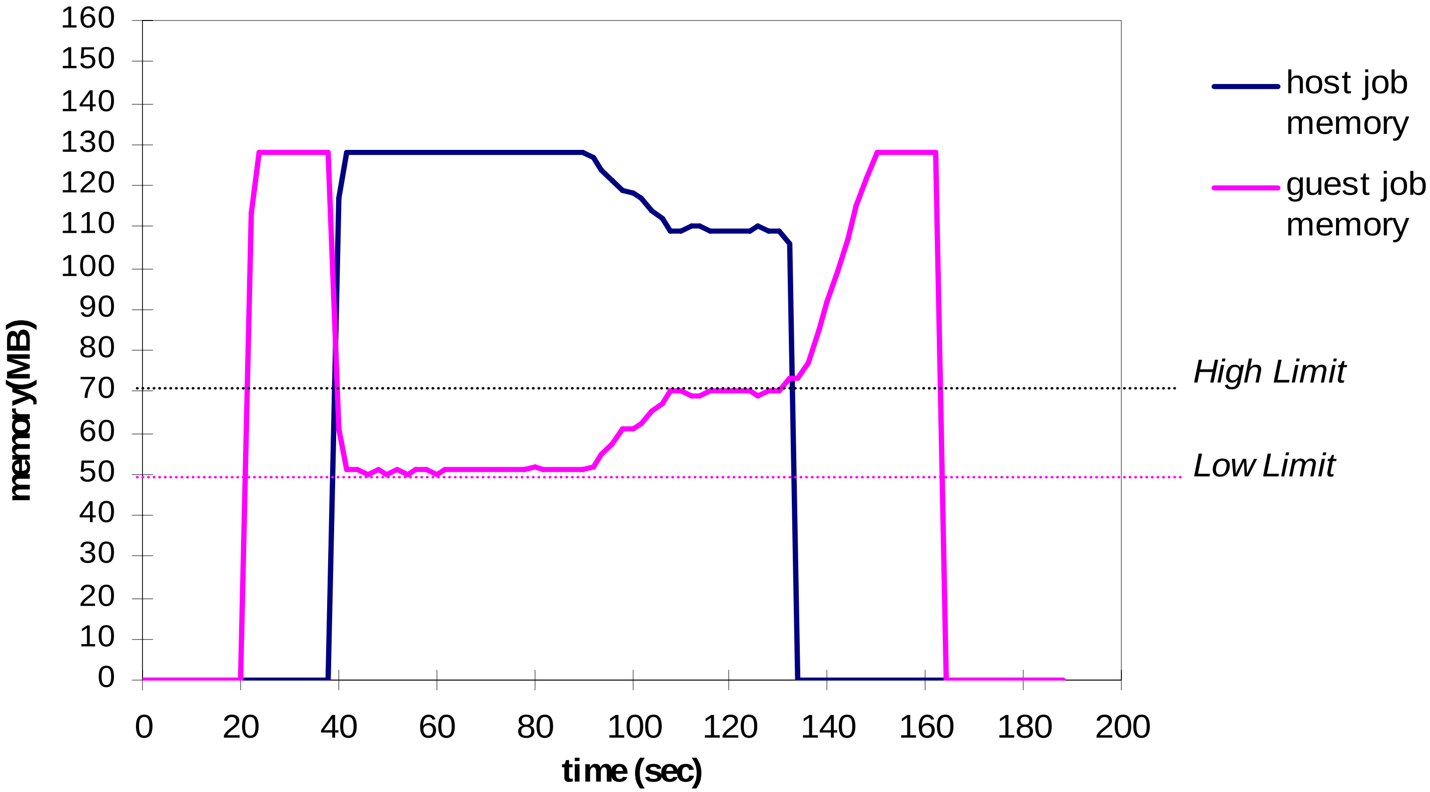


  - No limit on taking free pages
  - High Limit :
    - Maximum pages guest can hold
  - Low Limit :
    - Minimum pages guaranteed to guest

- Adaptive Page-Out Speed
  - When a host job steals a guest's page, page-out multiple pages
    - faster than default

Experiment: Memory Bounding

- Prioritized Memory Page Replacement
  - Total Available Memory : 180MB
  - Guest Memory Thresholds: High Limit (70MB), Low Limit (50MB)



Experiment: Nice vs. CPU & Memory Bounding

- Large Memory Footprint Job
  - Each job takes 82 sec to run in isolation

Policy and Setup	Host time (secs)	Guest time (secs)	Host Delay
Host starts then guest,			
Guest niced 19	89	176	8.0%
Linger priority	83	165	0.8%
Guest starts then host			
Guest niced 19	> 5 hours	> 5 hours	> 2,000%
Linger priority	99	255	8.1%

  - Host-then-guest:
    - Reduce host job delay 8% to 0.8%
  - Guest-then-host:
    - Nice causes memory thrashing
    - CPU & memory bounding serializes the execution

I/O and Network Throttling

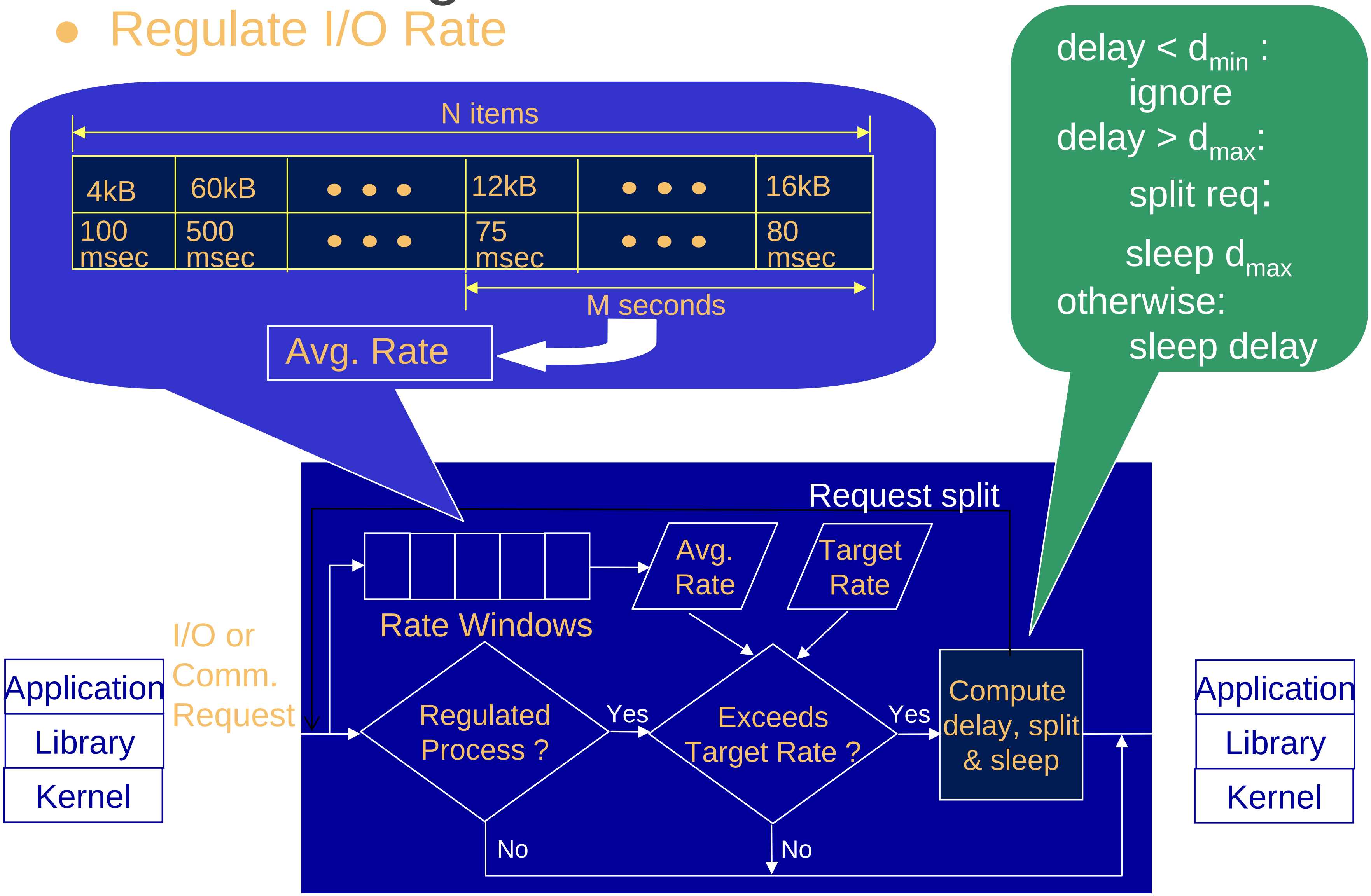
Problem 1: Guest I/O & comm. can slow down local jobs

Problem 2: Migration/checkpoint bothers local users

- Policy: Limit guest I/O and comm. bandwidth
  - Only when host I/O or communication is active
- Mechanism : Rate Windows
  - Keep track of I/O rate by host and guest
  - throttle guest I/O rate when host I/O is active
- Implementation: a loadable kernel module
  - Highly portable and deployable
  - Light-weight : I/O call intercept

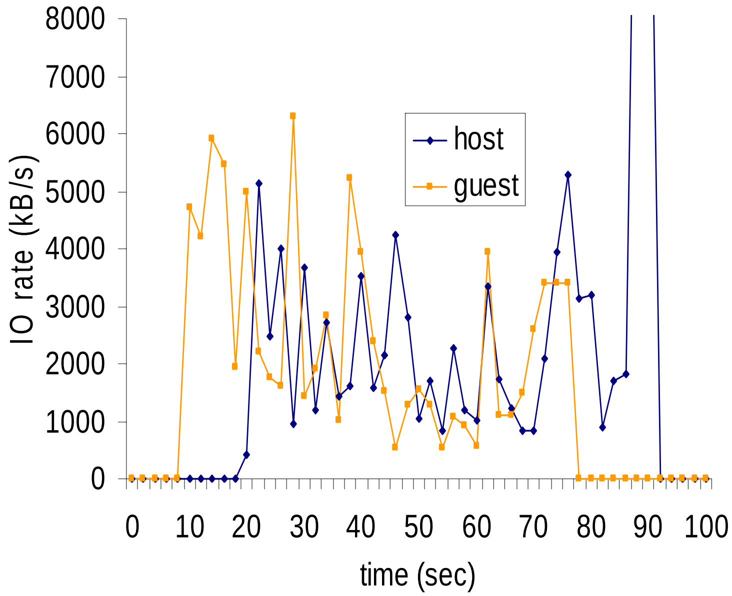
I/O Throttling Mechanism : Rate Windows

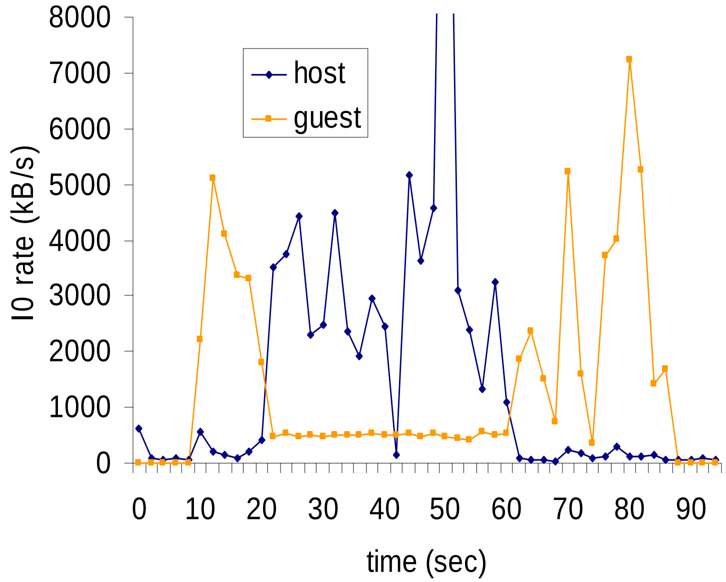
- Regulate I/O Rate



Experiment: I/O Throttling

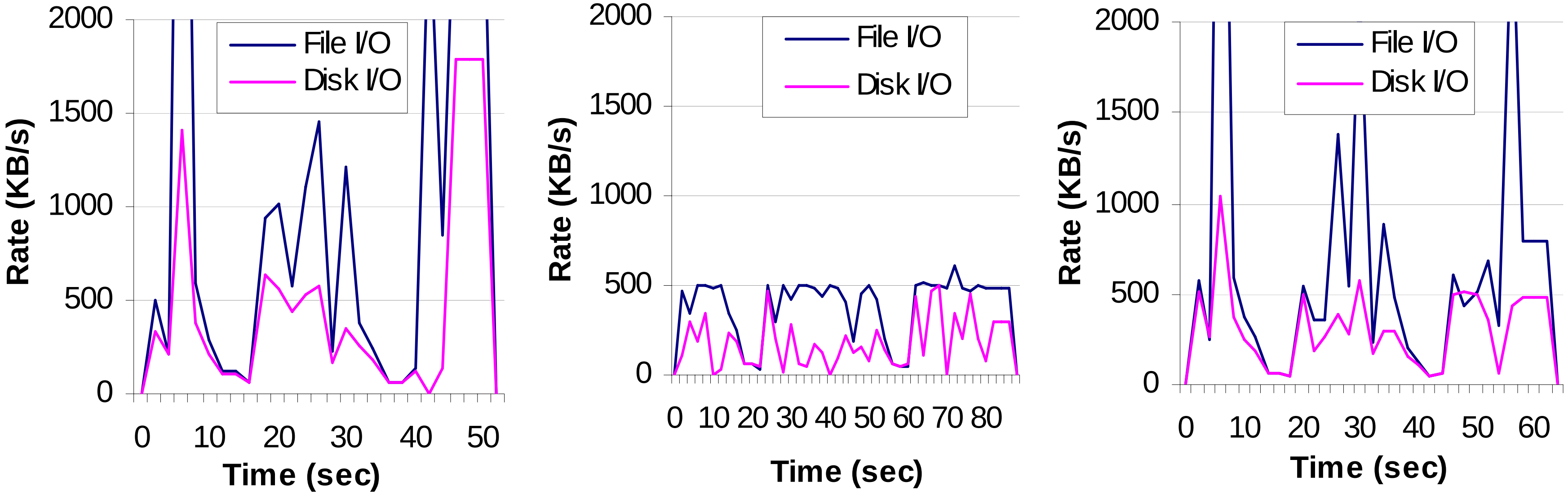
- tar programs as host and guest jobs
  - Guest I/O Limit : 500 kB/sec (~10%)
  - Throttling Threshold : Lo: 500 kB/sec Hi: 1000 kB/s





Dilation Factor in I/O Throttling

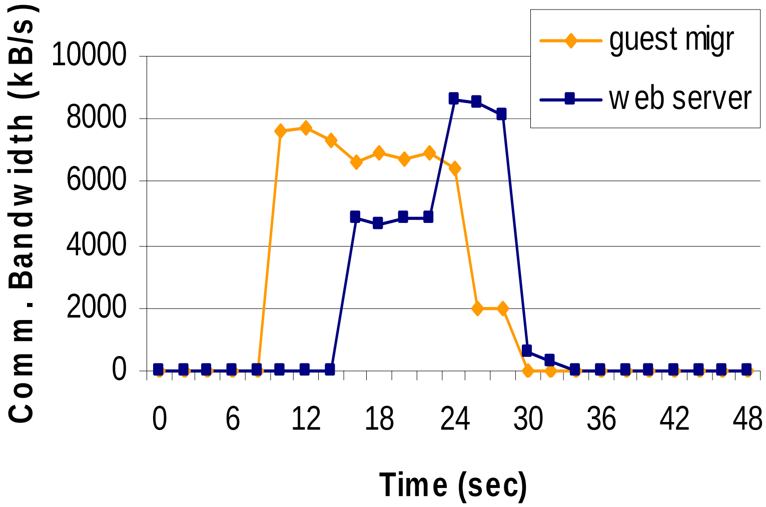
- File I/O rate ≠ disk I/O rate
  - Buffer Cache, Prefetching
- Control disk I/O by throttling file I/O
  - Adjust delay using
    - dilation factor = avg. disk I/O rate / avg. file I/O rate
  - Compile test (I/O Limit: 500kB/s)

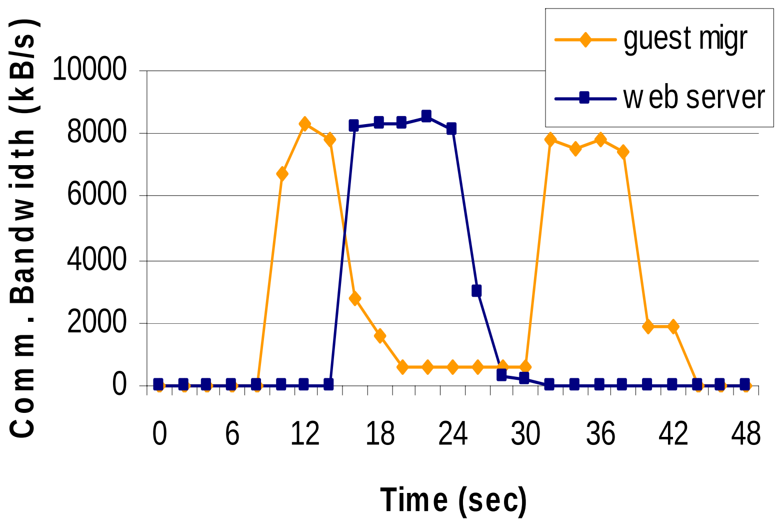


(a) No limit (b) File I/O limit (c) Disk I/O limit

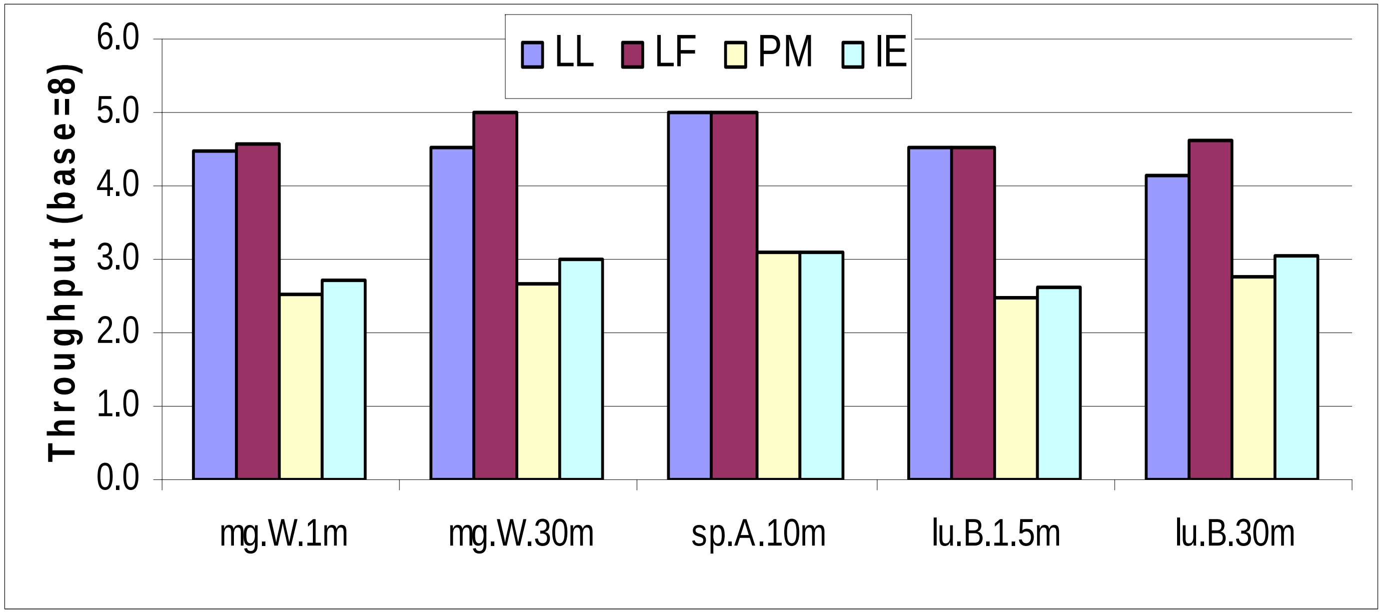
Experiment: Network Throttling

- Guest job migration vs. httpd as a host job
  - Guest job migration disrupts host job communication
  - Throttling migration when host job comm. is active
  - Guest job comm. Limit: 500 kB/s



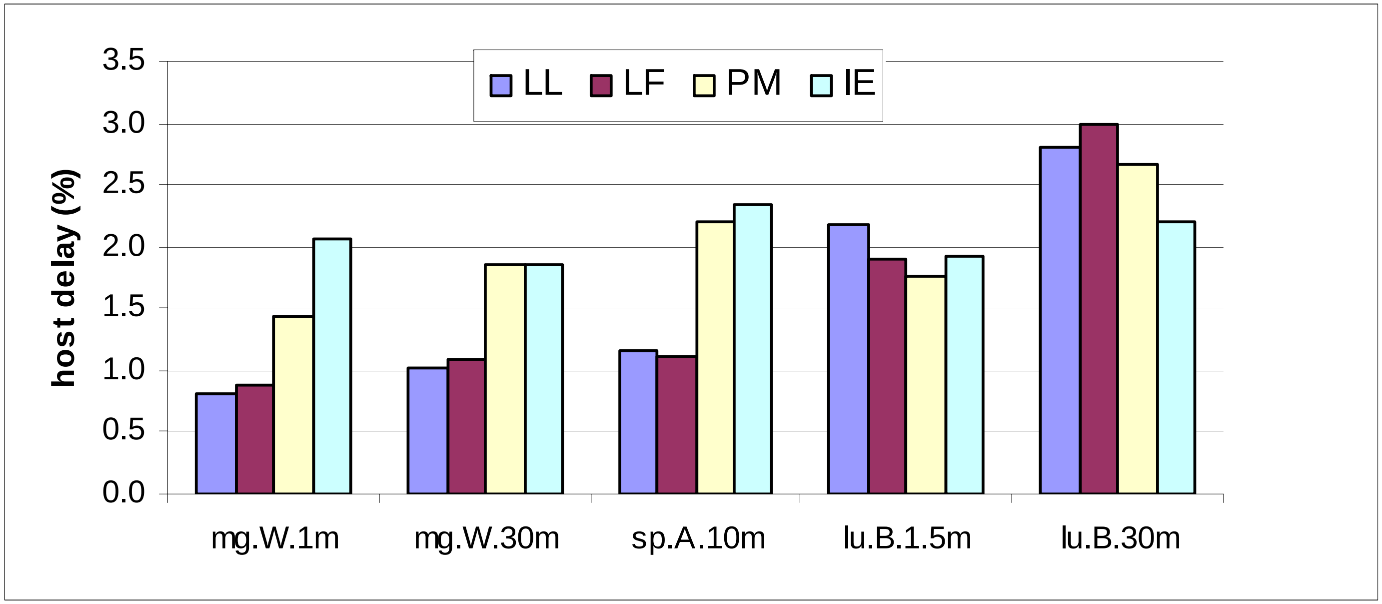


Guest Job Performance



- Overall, LL improves 50%~70% over IE
- Less improvement for larger jobs (lu.B)
  - Only 36% improvement for lu.B.30m
  - Less memory is available while non-idle
- LF is slightly better than LL
- Less Variation for LL
  - lu.B.30m: 23.6% for LL, 47.5% for LF

Host Job Slowdown



- LL/LF delays less for small and medium size jobs
  - 0.8%~1.1% for LL/LF, 1.4%~2.3% for PM/IE
  - Non-prioritized migration operations of PM/IE
- More delay for large jobs
  - Memory contention

Conclusions

- Identified opportunities for fine-grain idle resources
- Linger-Longer can exploit up to 60% more idle time
  - Fine-Grain Cycle Stealing
  - Adaptive Migration
- Linger-Longer can improve parallel applications in NOW
- A suite of mechanisms insulates local job's performance
  - CPU scheduling: starvation-level priority
  - Memory Priority: lower and upper limits
  - I/O and Network Bandwidth Throttling: Rate Windows
- Linger-Longer really improves
  - Guest job throughput by 50% to 70%
  - With a 3% host job slowdown

Related Work

- Idle Cycle Stealing Systems
  - Condor [Litzkow88]
  - NOW project [Anderson95]
  - Butler [Dannenberg85], LSF [Green93], DQS [Zhou93]
- Process Migration in OS
  - Sprite [Douglis 91], Mosix [Barak 95]
- Idle Memory Stealing Systems
  - Dodo [Acharya 99], GMS [Freely 95]
  - Cooperative Caching [Dahlin 94][Sarkar 96]
- Parallel Programs on Non-dedicated Workstations
  - Reconfiguration [Acharya 97]
  - MIST/MPVM [Clark 95], Silk-NOW [Brumofe 97]
  - CARMI [Pruyne 95] (Master-worker model)
- Performance Isolation
  - Eclipse [Bruno 98]
  - Resource container [Banga 99]