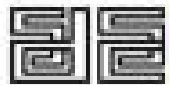


DGMonitor: a Performance Tool for Monitoring Resources on Sandbox-based Desktop Grids

M. Taufer, P. Cicotti, A. Chien



SDSC



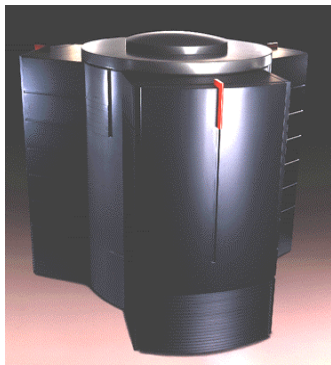
Search for More Cost-Effective Computing Resources

Late 80s

Middle 90s

Early 2000

time →



Cray C90



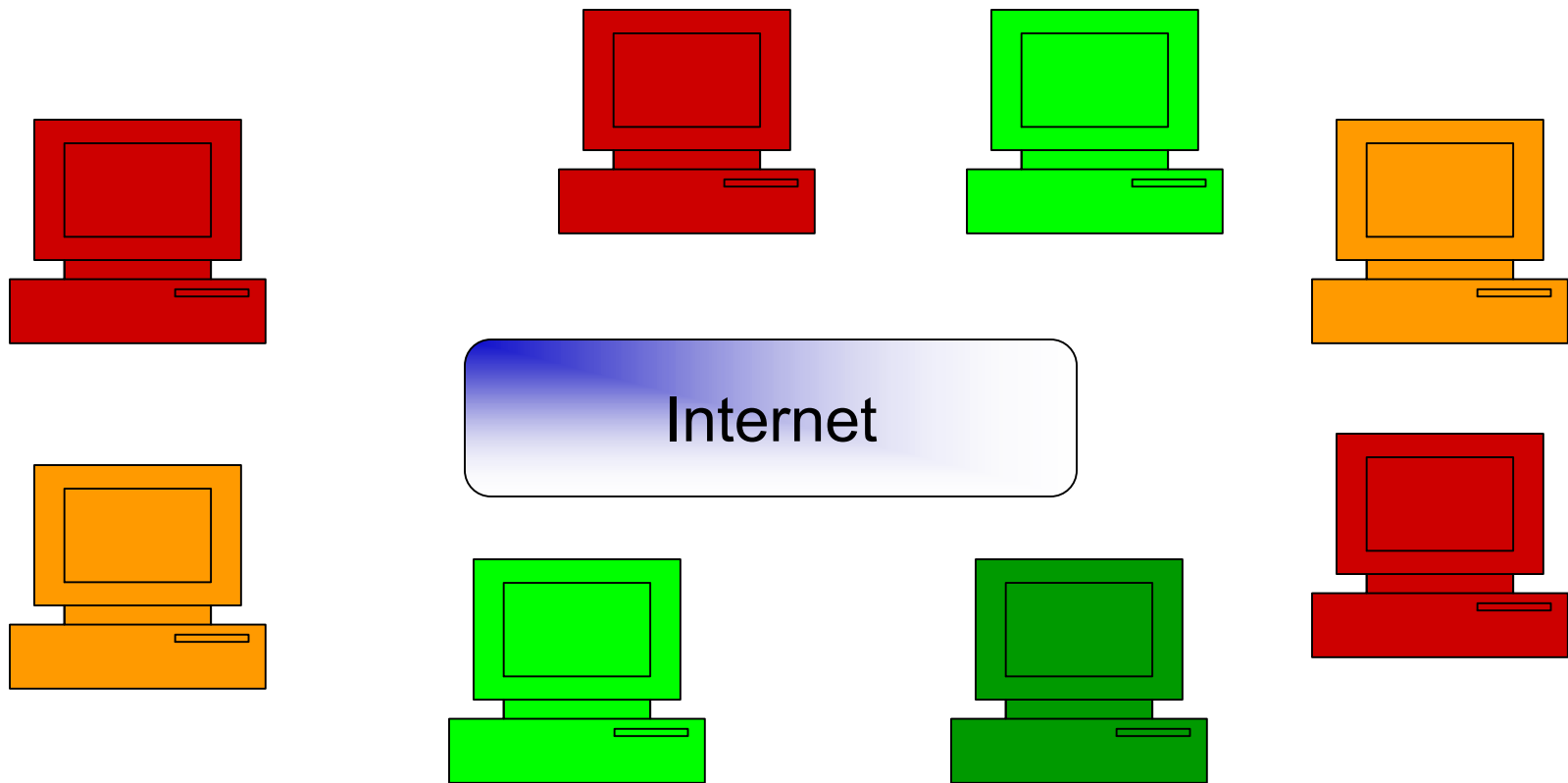
Cluster of PCs



Desktop grid

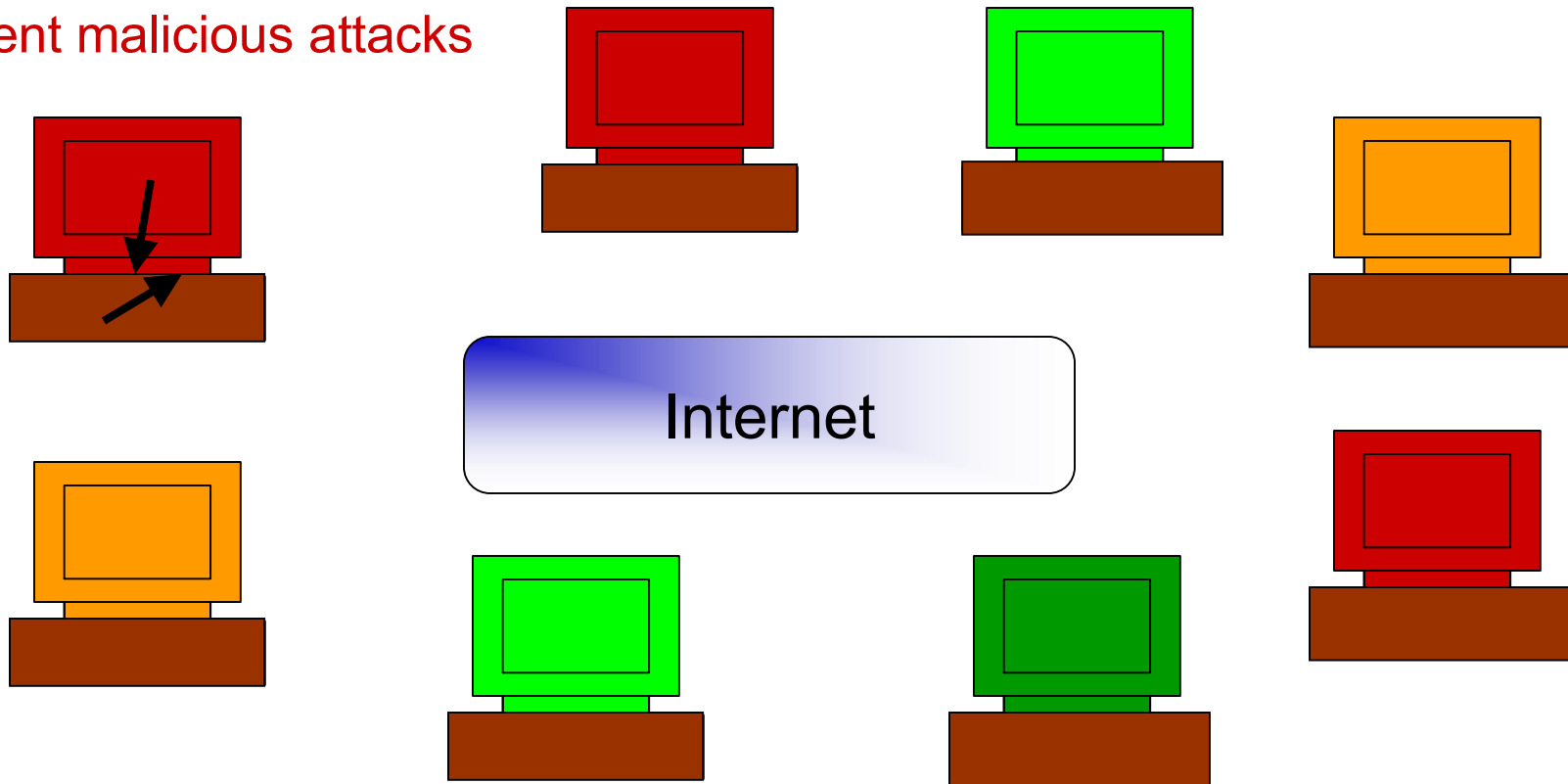
Desktop grids → By scavenging for available and idle cycles, they provide computing power at a significant cost savings

Heterogeneous and Volatile Computing Environment



Sandboxing Techniques

prevent malicious attacks



Challenge in Monitoring Sandboxed Applications

- For the sake of **performance** and **scalability**:
 - Discovery of resources among **volatile nodes**
 - Selection of resources based on **real information** at runtime
- On sandbox-based desktop grids, the **continuous monitoring of real resources** is a challenge:
 - Physical and virtual environments do not correspond
 - Sandboxes are reset at each task termination

DGMonitor as a tool for building an **accurate and continuous view of resource usage** for desktop grids based on sandboxing techniques

Outline

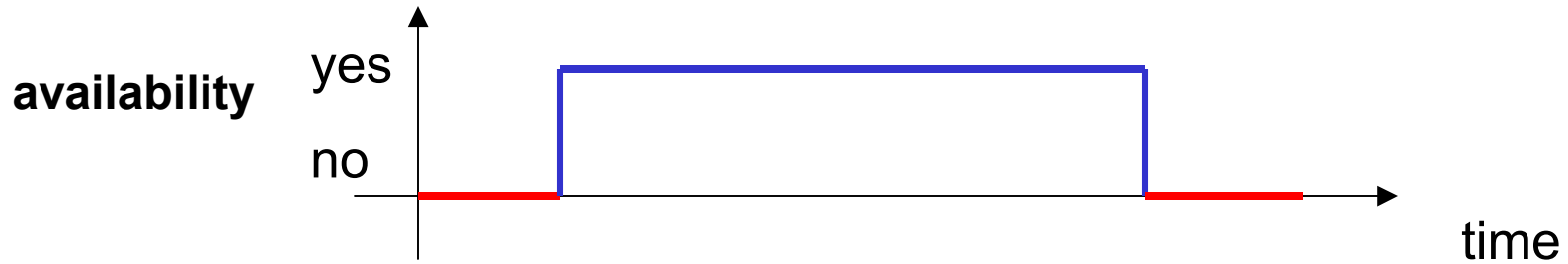
- Taxonomy of applications on desktop grids
- Monitoring resource availability
- Architecture of DGMonitor for Entropia desktop grid
- Critical implementation issues
- System evaluation
 - Local overhead
 - Scalability of DGMonitor
- Further research opportunities

Taxonomy of Applications on Desktop Grids

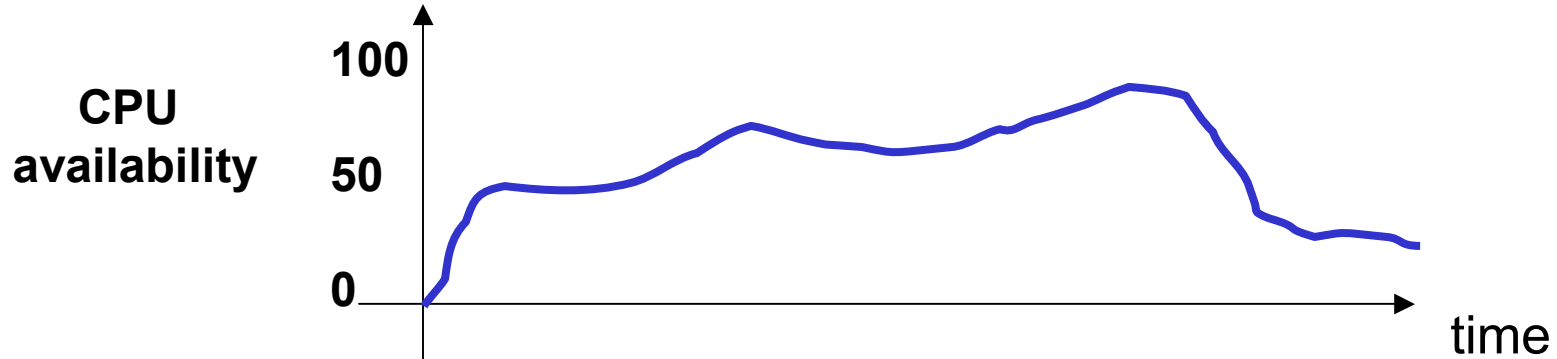
- Each application consists of a large number of independent tasks with **long turn-around time**
 - e.g., hours, days
- Total task execution can be decomposed into **distinct activities**
 - e.g., computation, communication, accessing data on local devices
- Each activity is largely characterized by the usage of one **single critical system resource**
 - e.g., CPU, disk, network

Monitoring Resources Usage

- Host availability:



- Resource availability (CPU, memory, etc.)



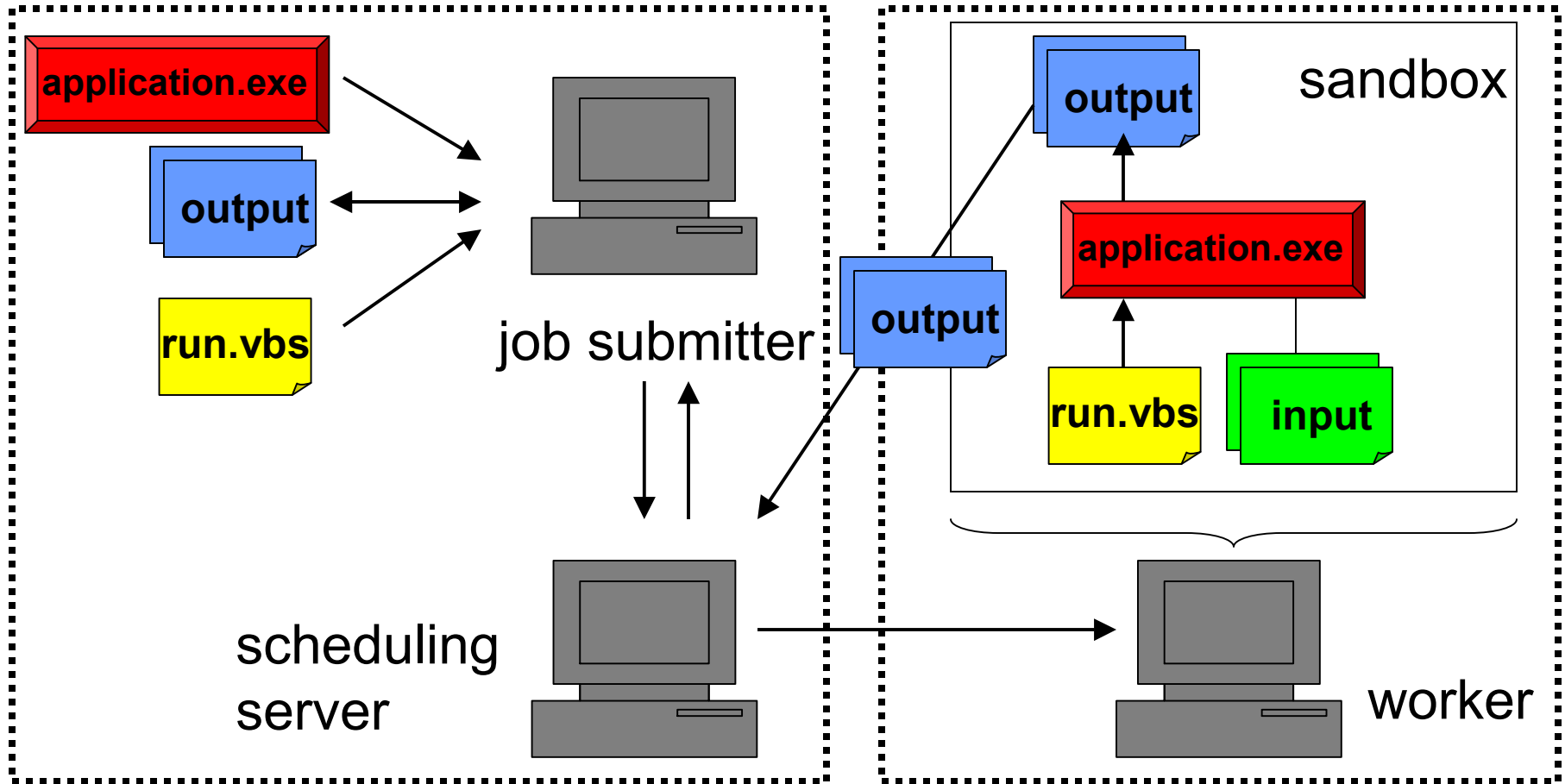
Advantages of DGMonitor

- No existing tools support desktop grids based on sandboxes as DGMonitor does
- Tools like VAMPIR, SVPablo, and Paradyn need:
 - Re-engineering, re-compiling or re-linking of the application
 - Embedded operating system or middleware instrumentation
- Tools like NWS or Ganglia require:
 - Handling sensors on local machines
 - Maintenance of two separate infrastructures → desktop grid platform and monitoring tool infrastructures

Entropy DCGrid

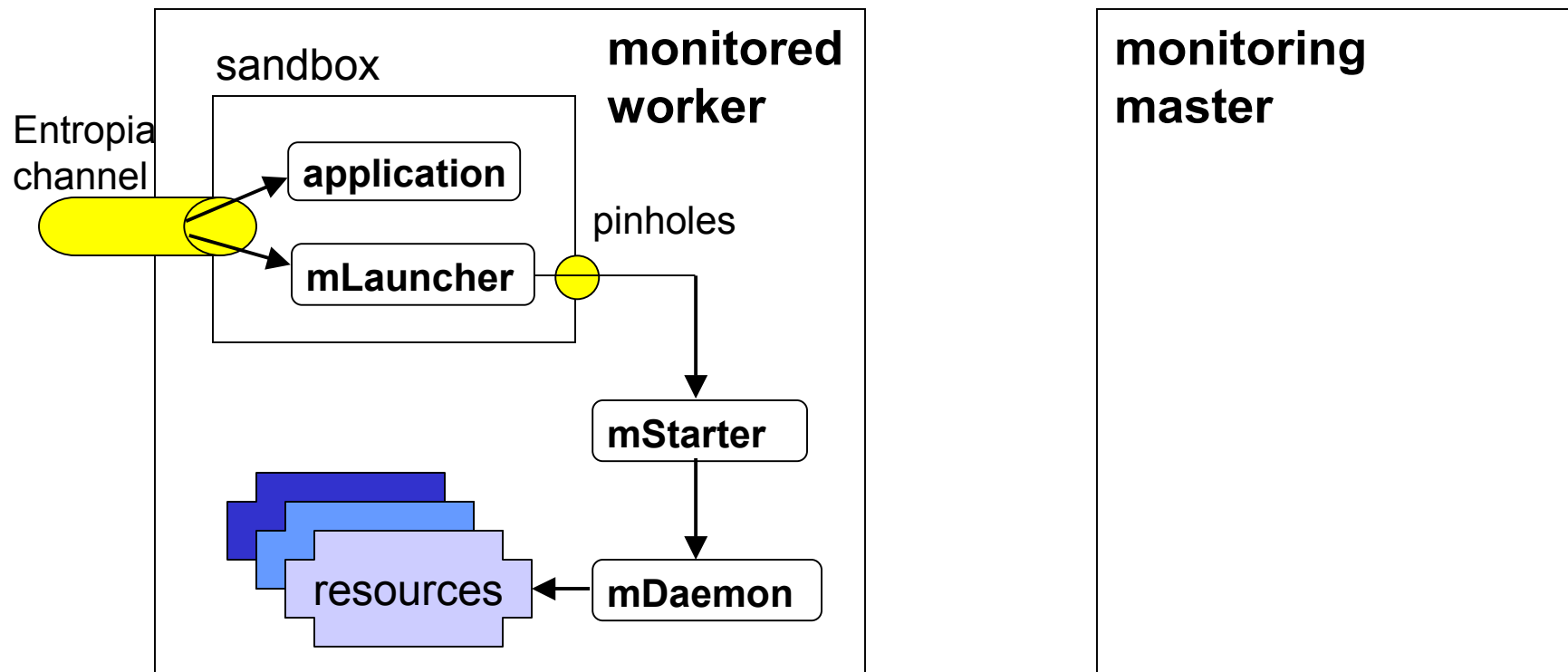
Job submission:

Job execution:



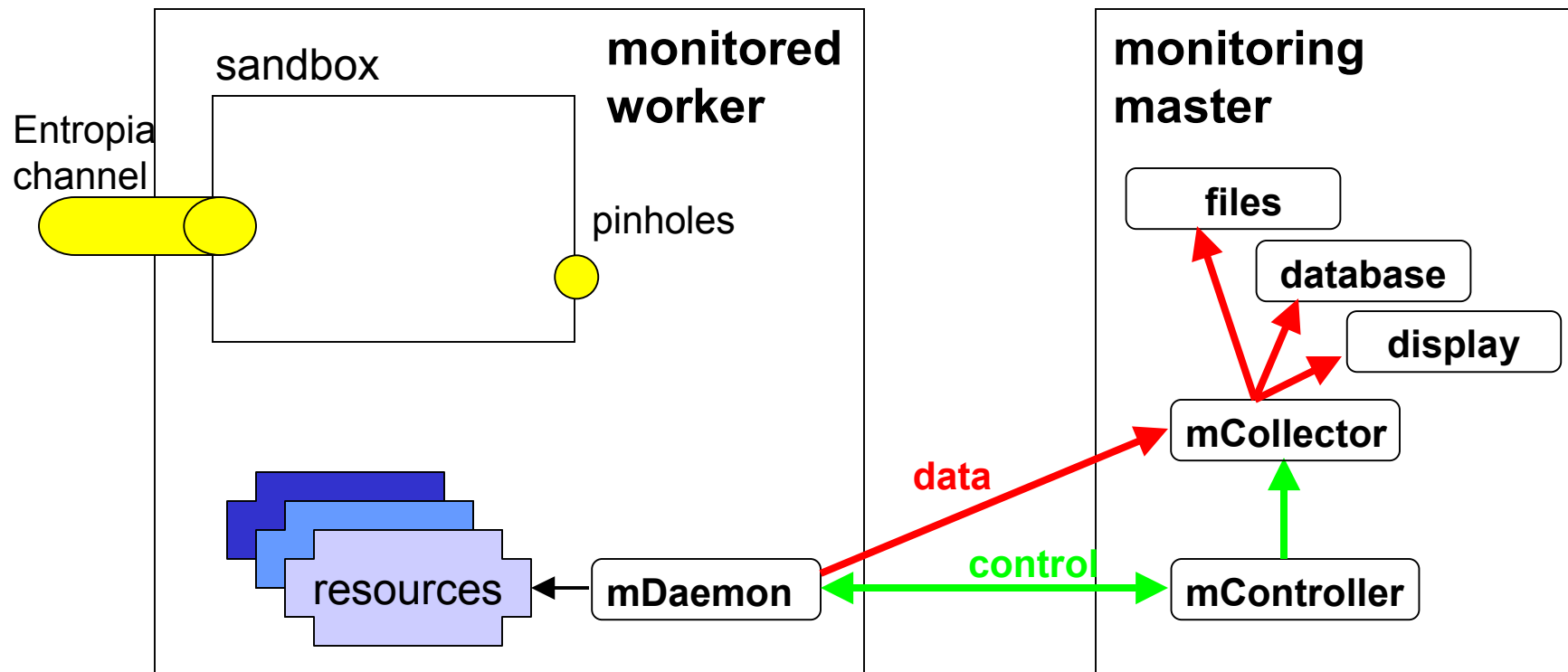
Architecture of DGMonitor

Our monitoring tool has a master-worker setting



Architecture of DGMonitor

Our monitoring tool has a master-worker setting



Critical Issues

- **Metrics:** what information to capture resource usage
- **Data sampling:** when to measure the status of resources
- **Data collection:** how to collect distributed information
- **Global notion of time:** how to fit distributed data in a global time framework
- **Monitoring intrusiveness:** how monitoring affects the monitored system / applications
- **Scalability:** what size for our monitoring tool



System Metrics

- Each desktop PC as a set of networked resources
 - i.e., CPU, memory, storage devices, network
- Each resource is represented by a set of metrics

Resources	Measured metrics
CPU	Idle time, user time, system time, number of processes, process queue length
Memory	Available cache, cache-, page faults page transferred
Disks	Transfer time, transfer/sec, transfer queue length, Byte/transfer
Network	Nic bandwidth, out queue length, packets transferred, packet dropped

Taxonomy of the Sampling

- Coarse granularity samples for long time simulations
- Daemon samples resources \vec{r} based on:
 - Time thresholds (δt , Δt with $1\text{sec} \leq \delta t \ll \Delta t$)
 - Resource thresholds (δr)



- *mDaemon* sends metrics to *mCollector* if:

$$\forall r_i \in \vec{r}(t): \quad |r_i(t) - r_i(t - \partial t)| \geq \partial r_i$$

- To facilitate the accurate rebuilding, samples are send every Δt
- Proper values for the time and resource thresholds depend on the system and the application

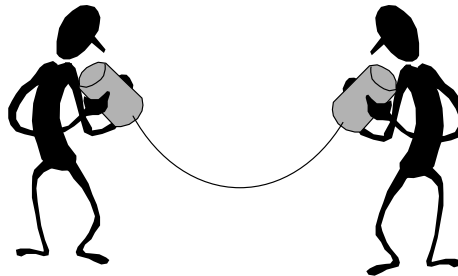
Communication Policy

Data channel

- *mDaemon* → *mCollector* : to send resource samples

Control channel

- *mDaemon* → *mController* : to register and keep the lease renewed
- *mDaemon* ← *mController* : to send time and resource thresholds, to renew lease



Communication Protocols

- Adverse effects of the monitoring data traffic by using the **UDP/IP protocol** for the collection of samples
 - Fast transmission protocol
 - No connection states are maintained → support for large number of workers
 - No attempt to re-transmit lost messages
 - Regression models to rebuild lost information
- Assure reliable control of the monitoring master over the workers by using the **TCP/IP protocol**
 - Control messages are not frequent

Ensuring Self-Cleaning by Using a Lease Policy

mDaemon survives the reset of sandbox **but** violates the obtrusiveness principle of sandboxing techniques

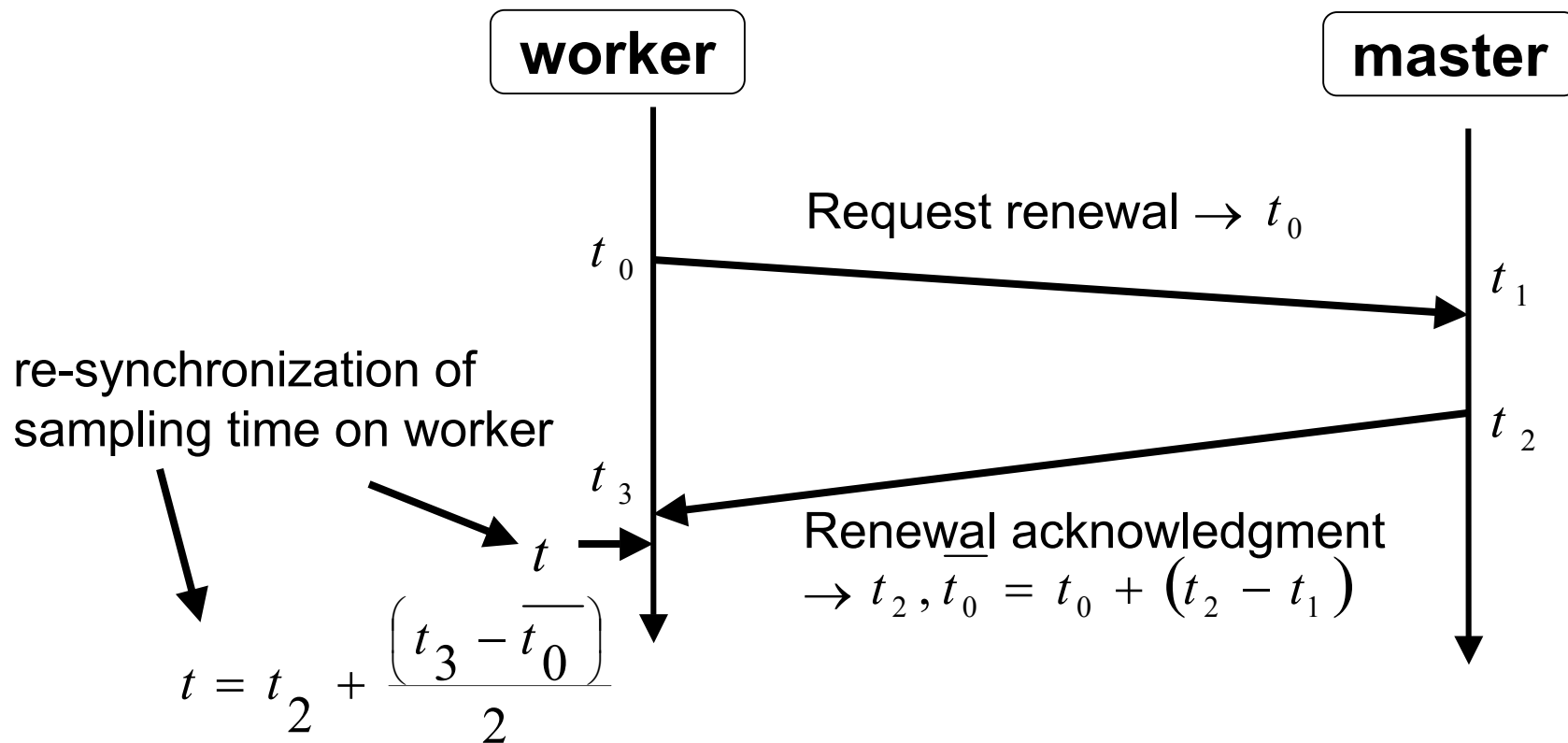
- At regular intervals ΔT , *mDaemon* requests renewal of its lease to *mController*
 - *mController* validates \rightarrow *mDaemon* continues its sampling
 - *mController* does not validate \rightarrow *mDaemon* terminates

Critical conditions:

- Network or *mController* failure \rightarrow *mDaemon* does not succeed in renewing the lease and expires
- *mDaemon* failure \rightarrow no attempt to resume *mDaemon*

Global Notion of Time

Notion of global time using virtual synchronizations based on Christian's algorithm



System Evaluation

System characterization

- Multi-threaded packet generator to emulate large number of monitored workers (performance data in packets of 128 bytes)
- Two different master configurations:
 - 400 MHz CPU, 256 MB memory, slower disk
 - 2 GHz CPU, 512 MB memory, faster disk

Local overhead

- Intrusiveness percentage on single worker due to monitoring

Scalability study

- Maximum number of *mDaemons* that *mCollector* can serve per second without losing performance data

Local Overhead

System characterization

- Different monitoring thresholds (δt , Δt , δr)
- Dedicated PCs

Measurement of intrusiveness on monitored workers

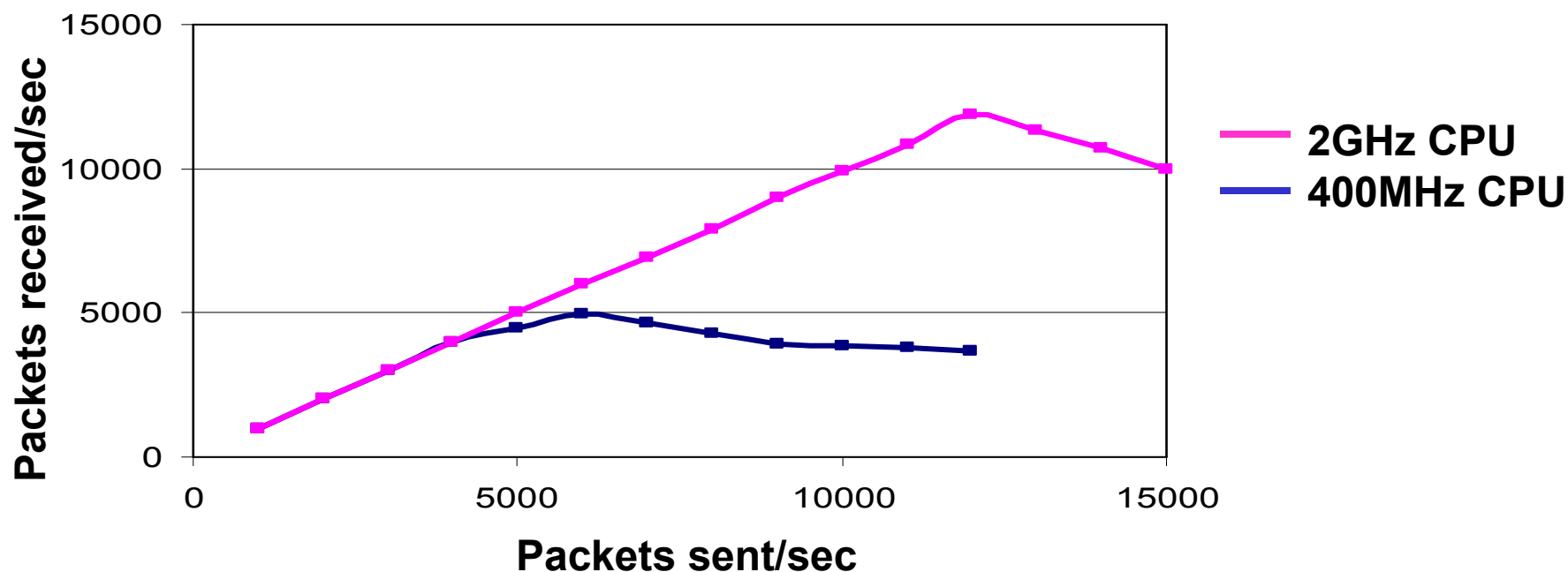
- The local overhead is **light and less than 1%** for repeated tests over long running times with:

$\delta t = \Delta t = 1 \text{ sec} \rightarrow$ maximal sampling frequency



Scalability for Different Master CPUs

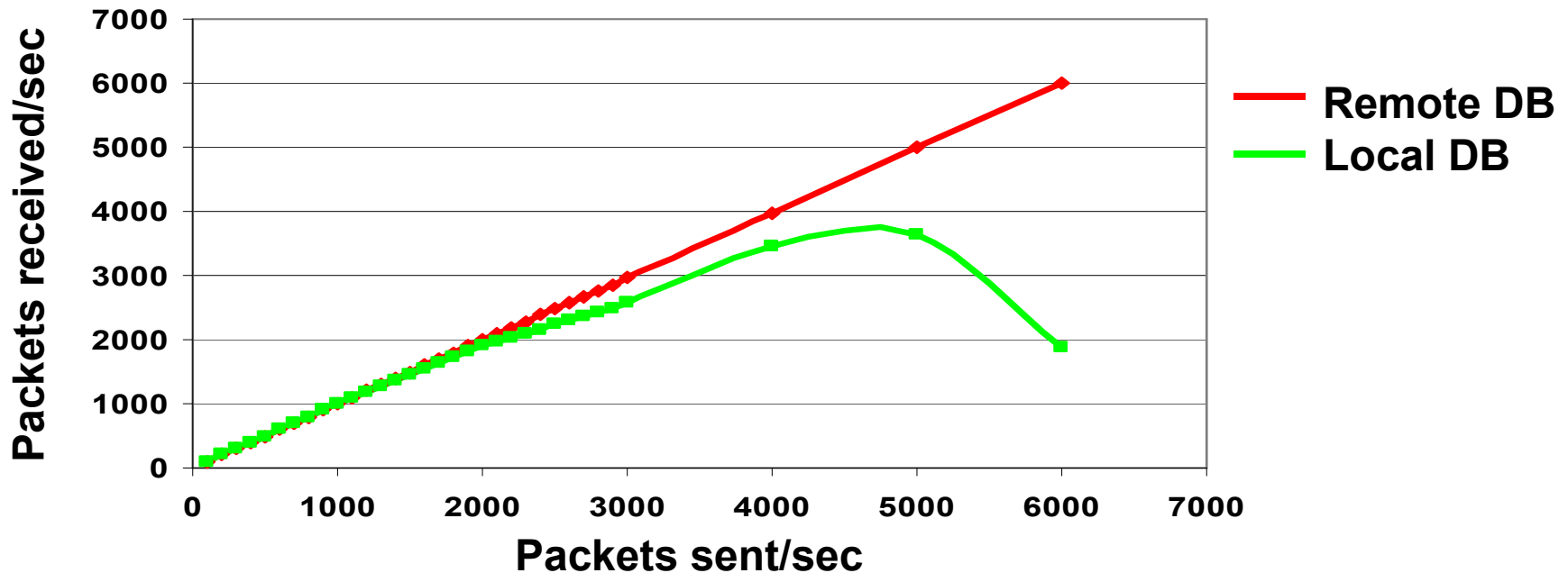
Collected performance data are saved in local files on the master:



The master can handle up to **12000 performance packets** per second sent in real time by independent workers

Scalability for Different Database Repositories

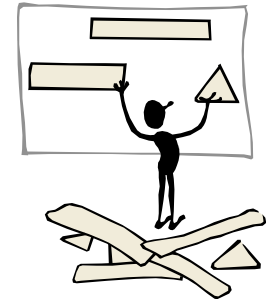
The faster master (2GHz) is used to collect performance data:



Remote DB provides **better scalability** but data are stored with some delay due to congestion control of TCP



Summary and Research Opportunities



- DGMonitor builds a **global, accurate, and continuous** view of real resource utilization for desktop grids with sandboxing
- DGMonitor scales to large desktop grids (**up to 12000 workers**) with low monitoring overhead (**less than 0.1% resource consumption** on monitored PCs)

Work on progress:

- Validation and accuracy of performance data collected
- Integration of DGMonitor into other desktop grid platforms i.e., XtremWeb
- Use of DGMonitor to address the resource requirements of applications at run time