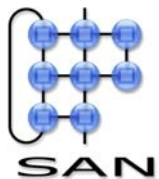


Concepts of Distributed Systems 2006/2007

Processes

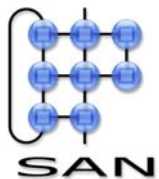
Johan Lukkien



Programme

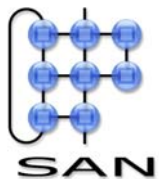
- Introduction & overview
- Communication
- Distributed OS & Processes
- Synchronization
- Security
- Consistency & replication

- Naming
- Fault Tolerance



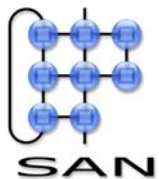
Processes

- A computer system typically realizes a few computational activities on some data
- At a given time, such an activity is found to be in a certain state
- The actual goal of a system is achieved through the cooperation of several of these activities
- This presentation is about how these activities are dealt with in a distributed context



Processes (in distributed context)

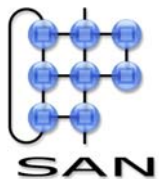
- Threads & processes
- Client & Server architecture
- Migration
- Agents
- Example: load balancing



Threads & Processes

- Process (“program in execution”)
 - defines a data space
 - virtualization
 - has at least one associated thread
 - unit of distribution

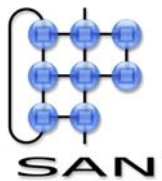
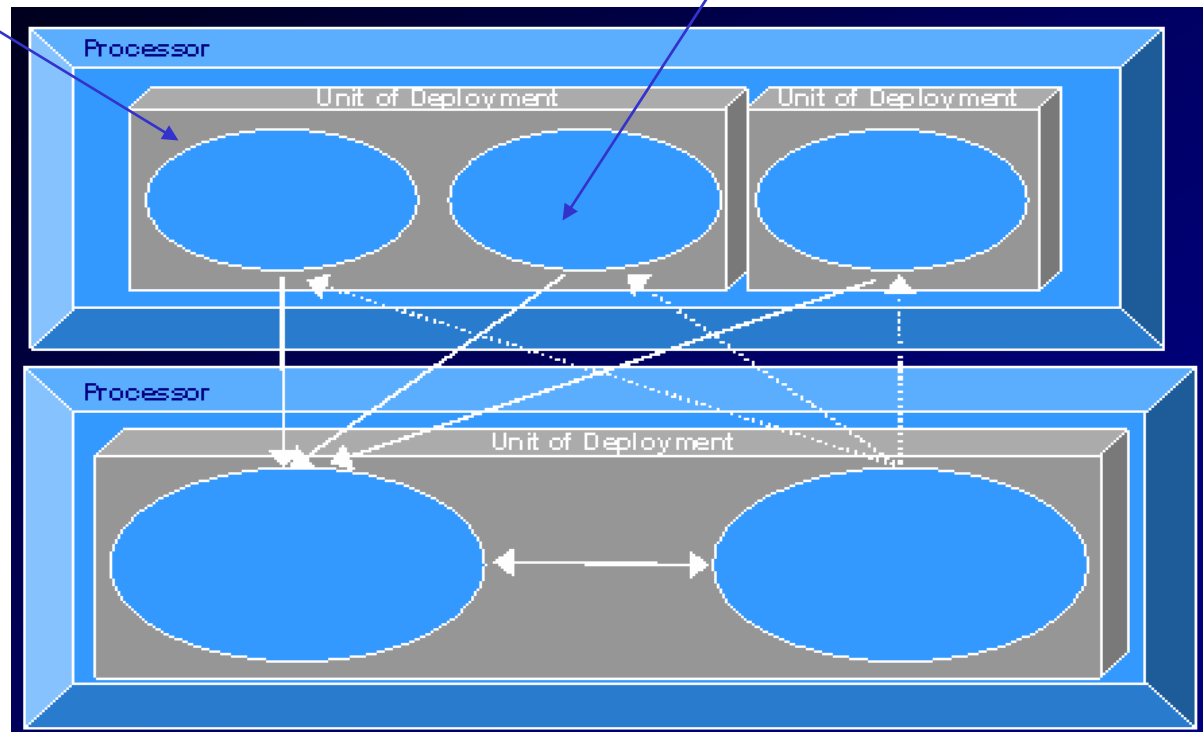
- Thread
 - unit of concurrency
 - unit of scheduling
 - though with one thread per process, a process is often said to be the unit of scheduling
 - several threads ‘live’ within shared address space
 - i.e., in a process



Schematic

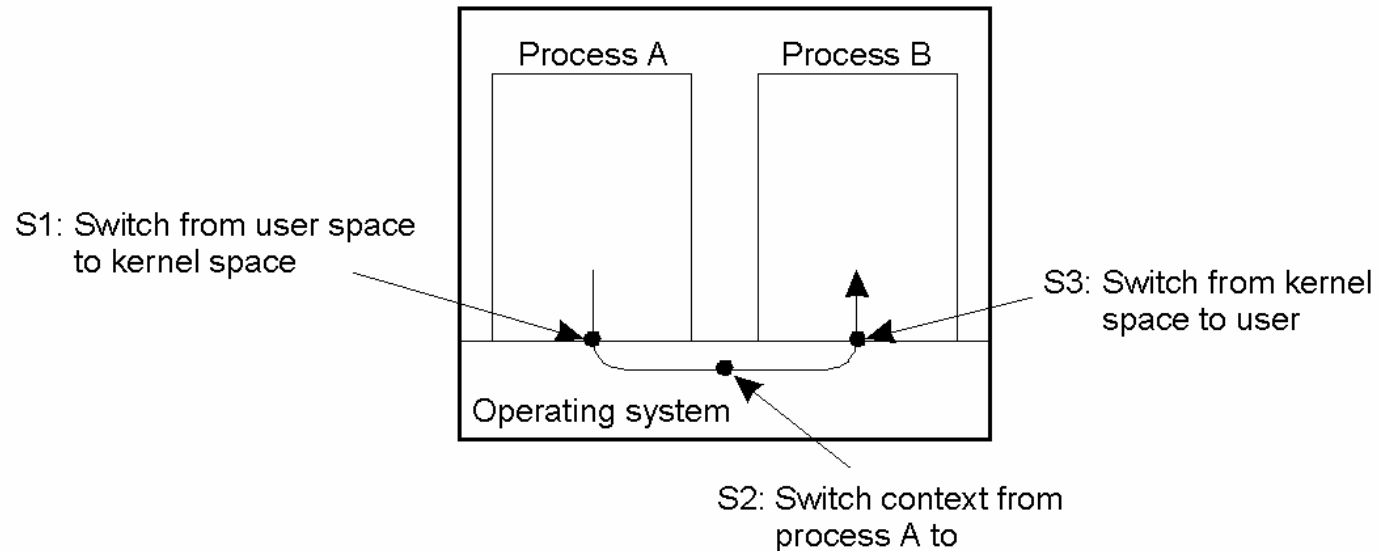
Process

Thread



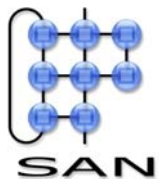
Process overhead

- Creation
- Switching
 - for each Inter Process Call – no shared memory
 - TLB, MMU



Threads

- Concurrency on shared memory
 - no protection
 - no expensive IPCs between threads
 - easier to construct programs (?)
 - exploit platform concurrency
 - latency hiding
- State per thread
 - control state: `context`
 - stack
- Generally, little overhead for switching
 - depending on underlying execution model

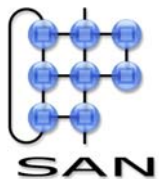


Mode and context

- A mode switch
 - changes the processor mode and loads a new program counter
 - is needed for any event that requires kernel activity
 - must usually be followed by saving some additional processor state

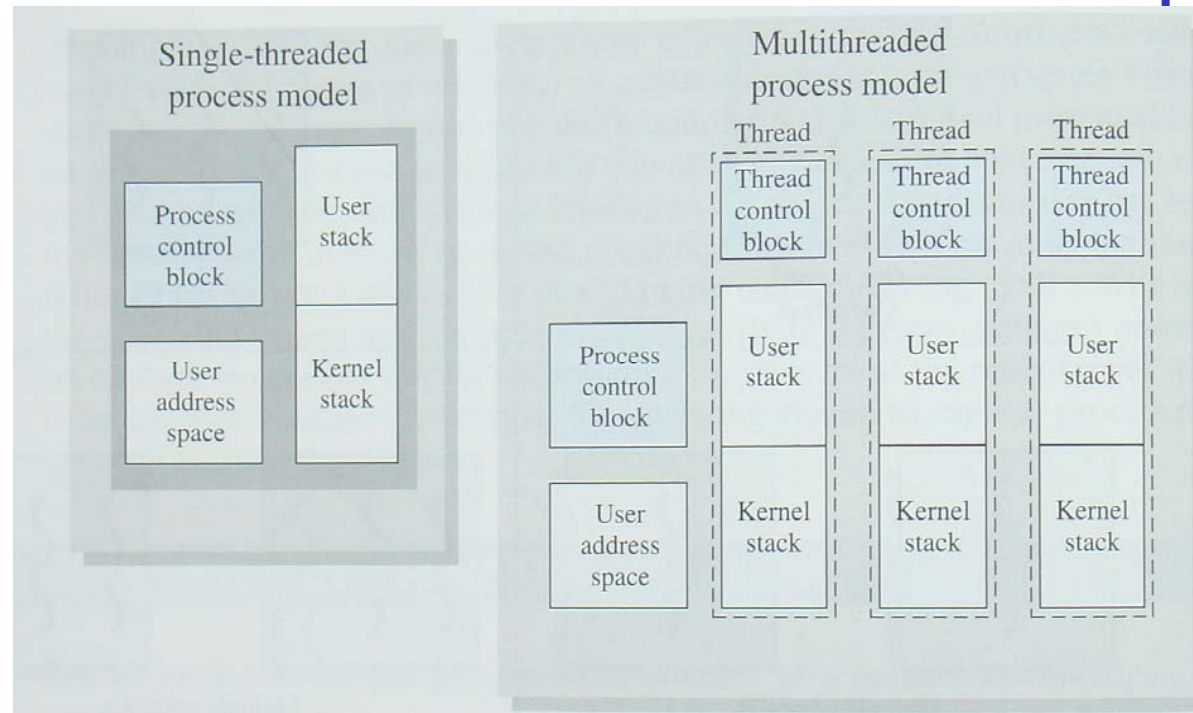
- A context switch
 - changes the memory management settings
 - is needed, besides a mode switch, *for only some events*, e.g. if a process switch is needed

- Except for the handler, most of the overhead of a mode switch is performed in hardware
 - hence, we would like to restrict the majority of the event handling to just this



Single and multi-threading

- Notes:
 - a thread-switch can be realized just through a mode switch
 - light-weight process
 - threads that share an LWP (i.e. user-level threads):
 - share a control block
 - don't need a mode switch



from: W. Stallings, *Operating Systems, Internals and Design principles*

POSIX: Thread interface

- Start a function as a new thread
 - setting attributes, e.g. stacksize;
 - thread terminates when this function returns
- Extensive interface
 - detach, join, cancel

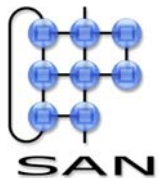
```
pthread_t thread_id;
```

```
status = pthread_create (&thread_id, attr, func, arg_of_func);
```

```
/* func(arg_of_func) is started as a new thread; when attr == NULL some  
* internal defaults are chose */
```

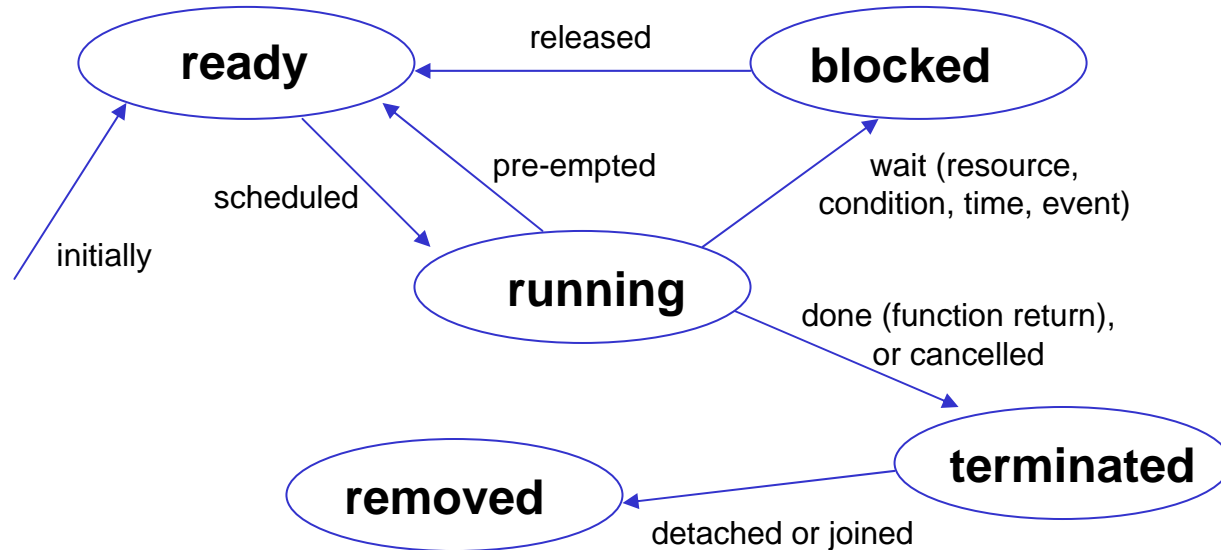
```
status = pthread_join (thread_id, &result);
```

```
/* wait for thread to terminate */
```



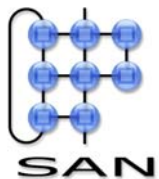
The life of a thread in POSIX

- Main program: own thread
- Additional threads:
 - created on demand or on receipt of signal
 - thread code: a function in the program



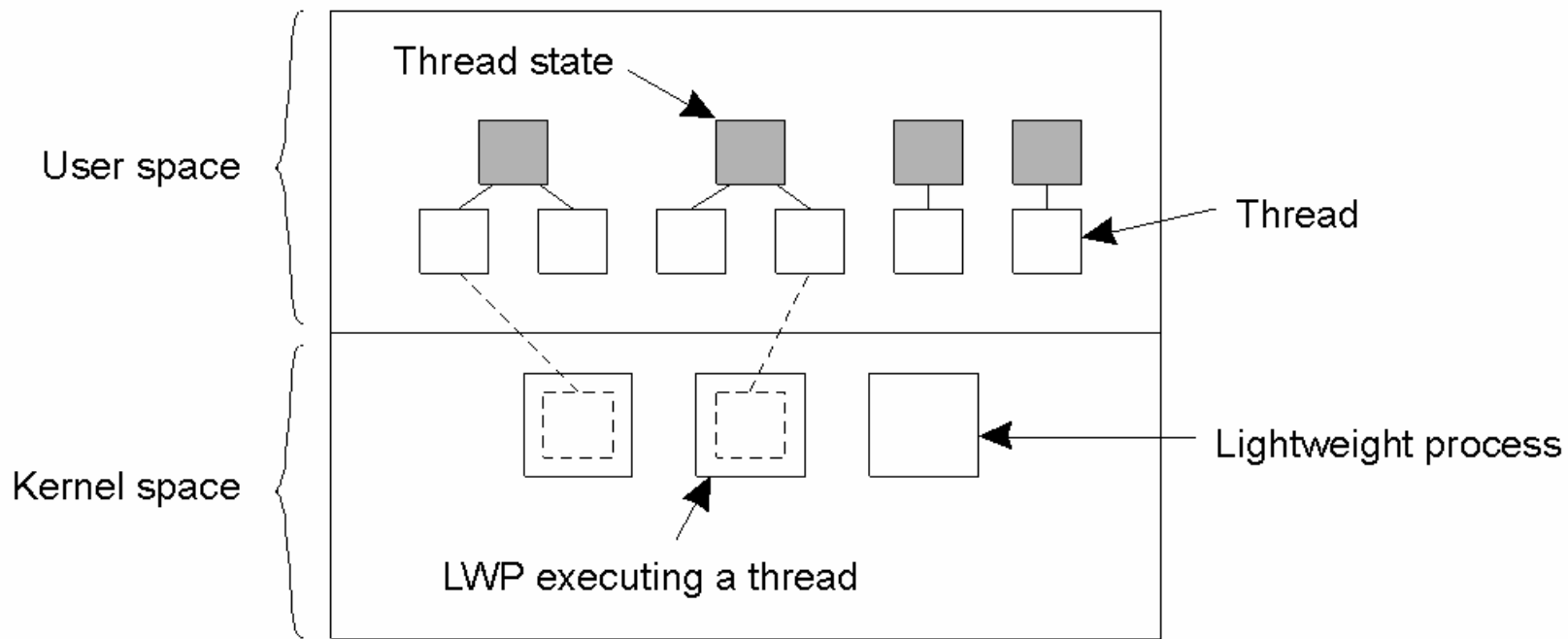
Thread execution model

- **Concurrency level:** number of “engines” (virtual processors) actually executing the threaded program
- Virtual processors are scheduled by the kernel
- Concurrency level choices
 - 1: no concurrency, no kernel activity in switching (all threads are ‘user-level’ threads)
 - # threads: switching always becomes kernel activity
 - in between: only blocking kernel calls and virtual processor scheduling requires kernel activity



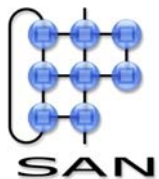
Thread execution: Solaris

- Virtual processor: lightweight process
 - LWP just executes threads (no memory space)
 - blocking calls in user space switch LWP to new thread



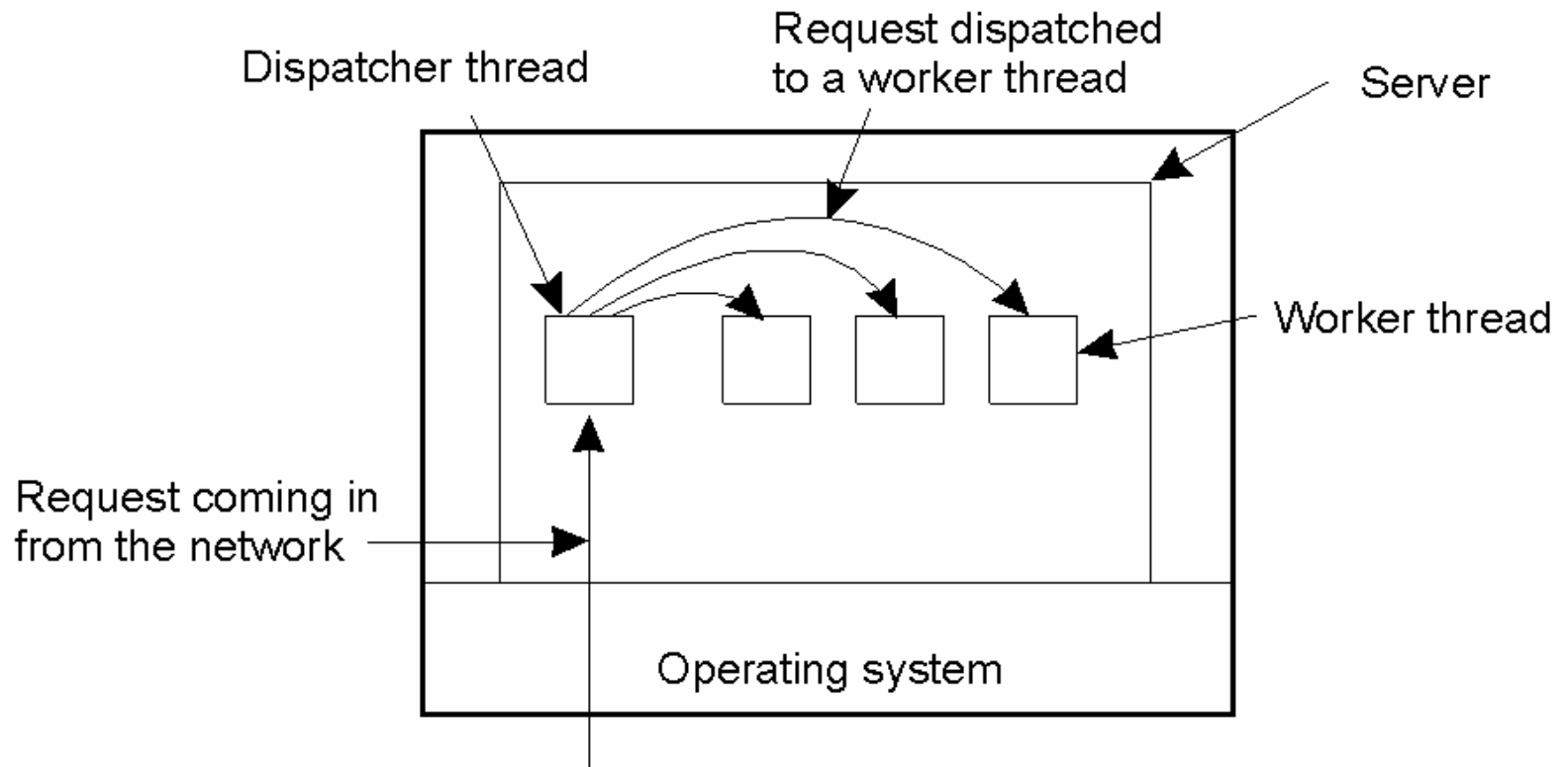
Thread use

- Client
 - hide latencies
 - communicate with several partners
 - all references on a Web page, while displaying
 - multiple RPC's
 -ease of programming
- Server
 - handle multiple clients
 - easy extension to multi-processor box
 - hide blocking services (e.g. disk access)
 - ...ease of programming



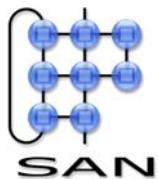
Multithreaded Servers

- Dispatch/worker model



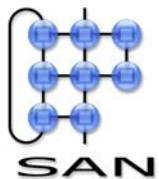
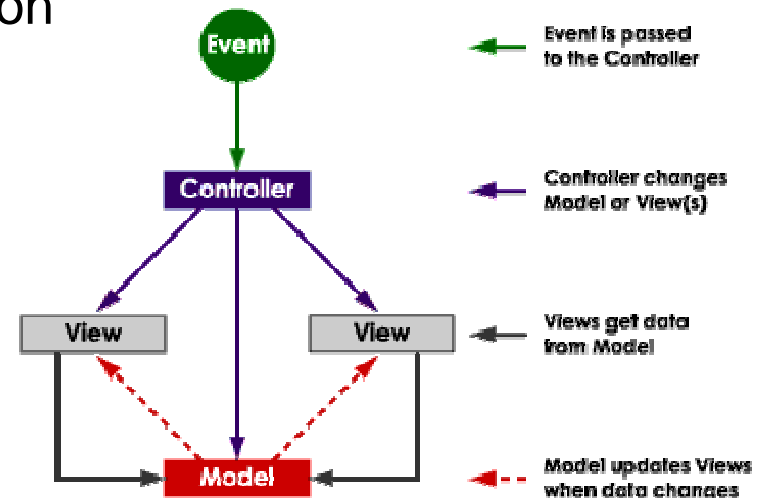
Processes (in distributed context)

- Threads & processes
- Client & Server architecture
- Migration
- Agents
- Example: load balancing

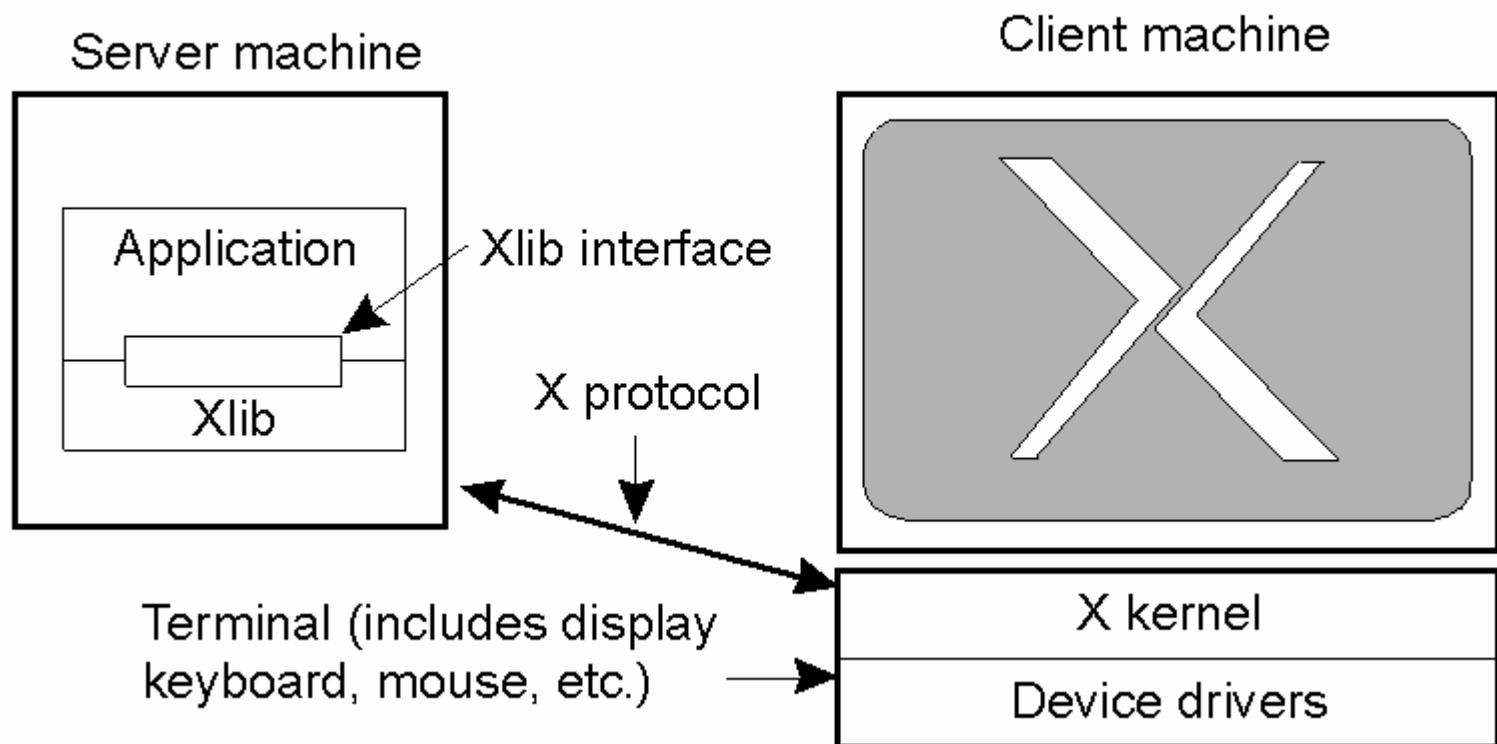


Model-View-Controller architecture

- Several views on same model ('data')
- User changes model by interaction with a view; events go to controller
- Connect controllers of application
 - standardize on user interaction
 - drag & drop
 - in-place editing
- Note: all arrows could be distributed

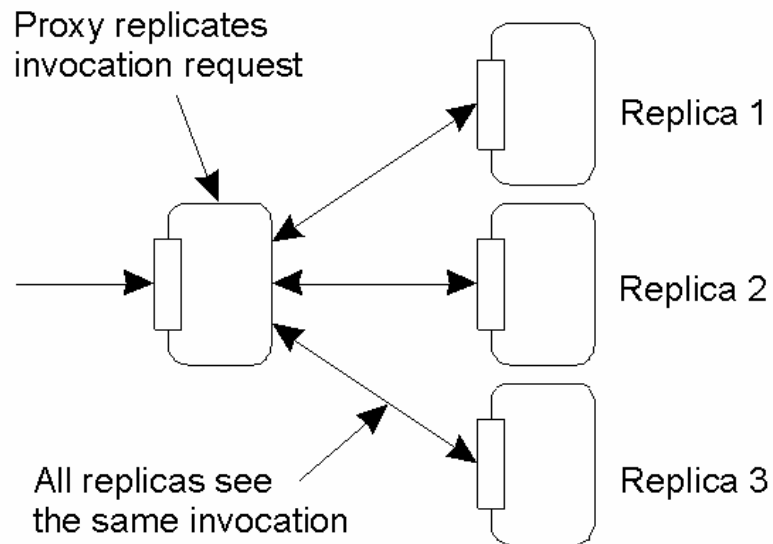


The X-Window System



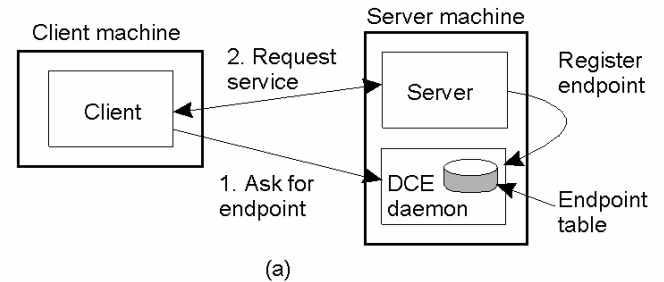
Transparencies at client

- Access, location, replication
 - in stubs, proxies, adapters

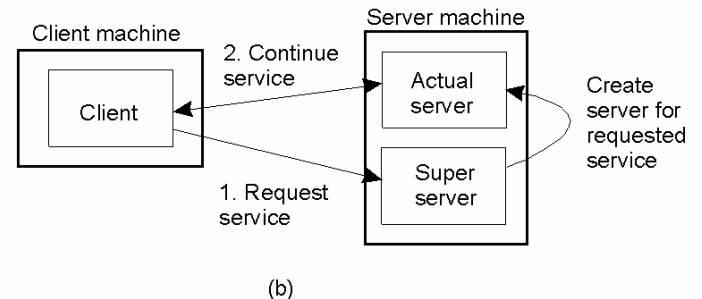


Server design

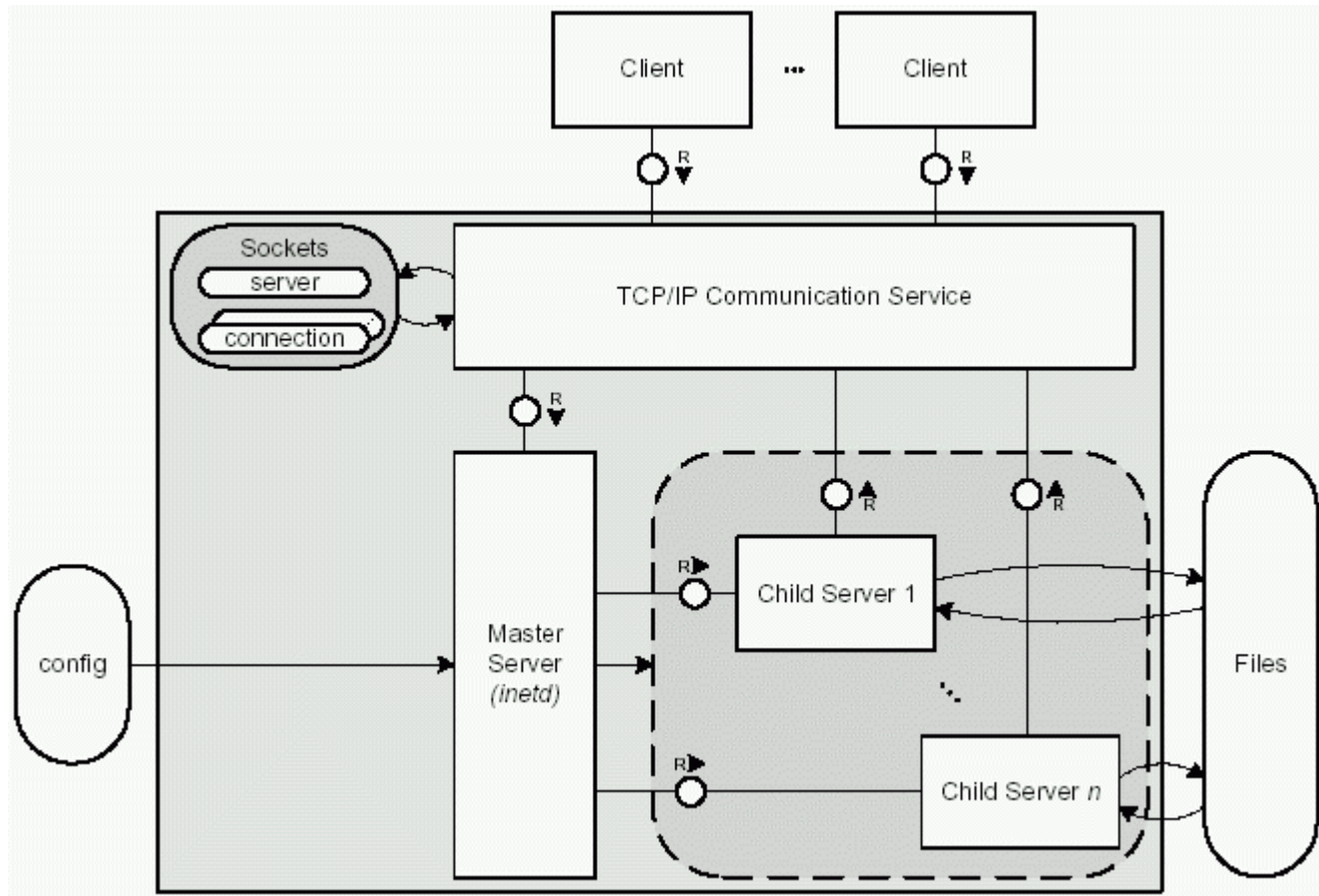
- Contact
 - Service coupled to fixed transport port
 - e.g., ftp: 21, telnet: 23 etc.
 - ‘Superserver’: handle many ports
 - dispatcher, e.g., inetd (at process level)



- How to interrupt server?
 - out-of-band communication
 - control connection
 - e.g. SIP, RTP



Inetd (fork upon request)

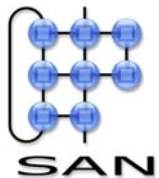


Picture from 'multitasking server architecture',
Grone, Knopfel, Kugel, Schmidt, Potsdam University

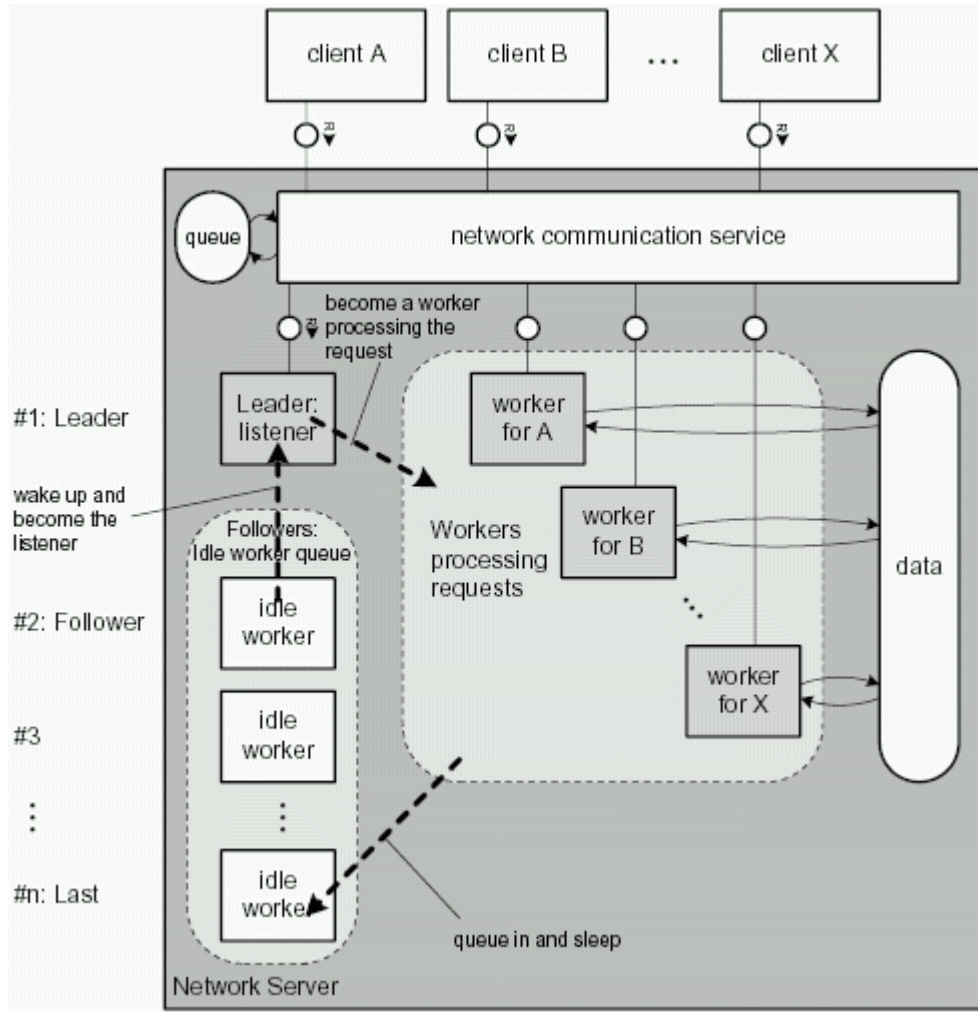
Server design

- Internal concurrency
 - iterative: single request/reply
 - concurrent: pass on request
 - internal multithreading
 - distributed organization

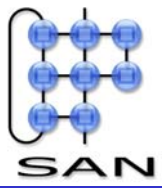
- Client state
 - Server side
 - preferred: stateless (open files, caches,..)
 - performance, complexity, scalability (1 Mb for 1000 clients is 1Gb)
 - Client side
 - cookies, web-page content



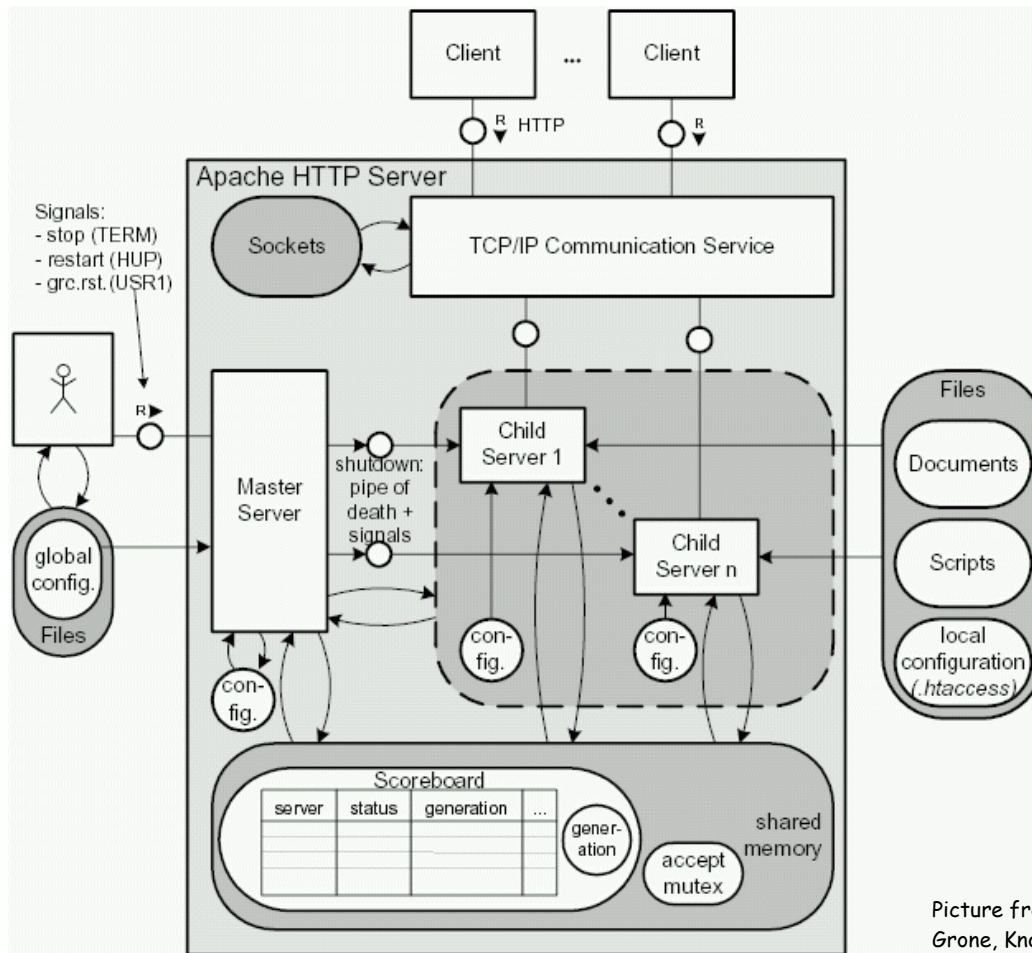
Workpool with sockets



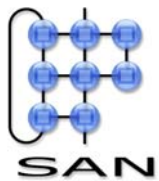
Picture from 'multitasking server architecture', Grone, Knopf, Kugel, Schmidt, Potsdam University



Apache organization (preforking)

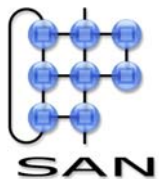


Picture from 'multitasking server architecture', Grone, Knopfel, Kugel, Schmidt, Potsdam University

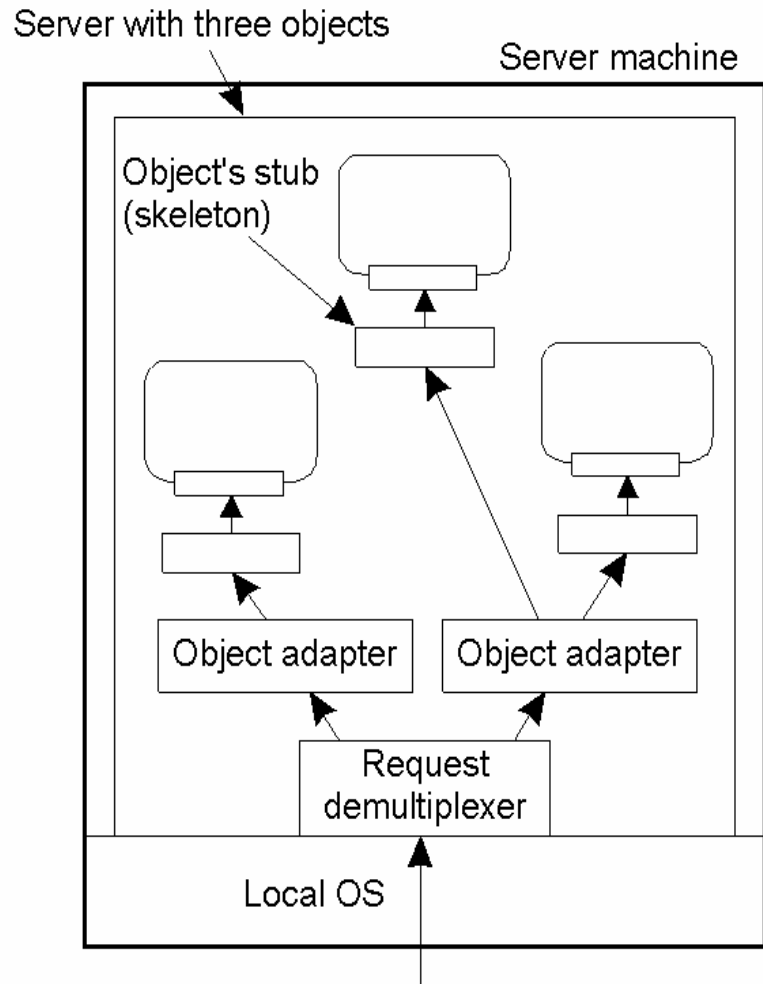


Object servers

- ‘Connect object to network’
- Policies:
 - transient
 - create upon initialization/1st access
 - destroy upon server down/last client away
 - memory sharing
 - threading
 - per object?
 - access & activation
 - skeleton: (un)marshalling, actual call
 - adapter: implements policy; registers object
 - request (de)multiplexing

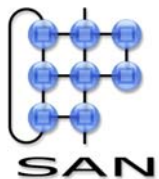


Example Object server



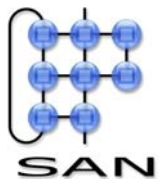
Processes (in distributed context)

- Threads & processes
- Client & Server architecture
- Migration
- Agents
- Example: load balancing



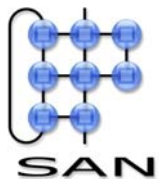
Migration

- Performance: load balancing
 - minimize job latency
 - **notice:**
 - maximixing resource utilization and minimizing job latency are conflicting
- Locality:
 - run code where data is
 - multiply code over data sources
- Postpone decisions
 - dynamic configuration
 - late binding



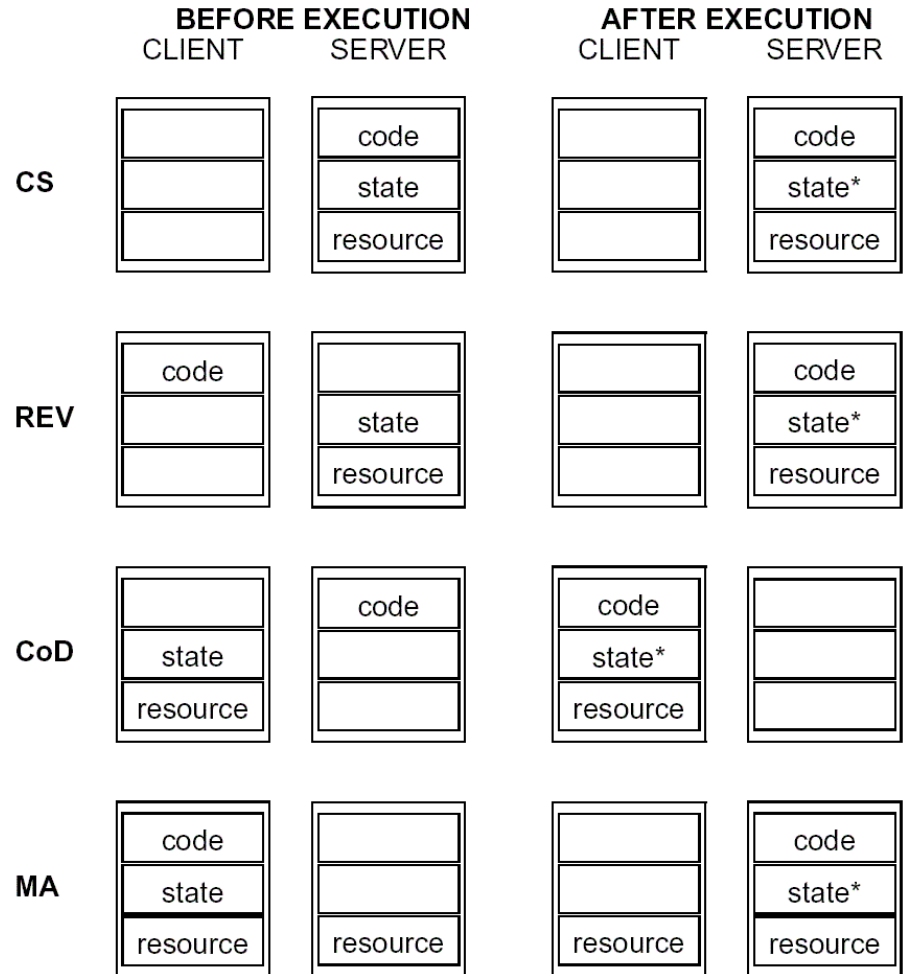
Migration framework (Fugetta)

- *Code and state using resources*
- State
 - execution segment
 - data
- Resources, affected by migration
 - memory
 - processing



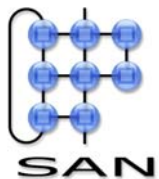
Examples

- CS: client-server
- REV: remote evaluation
- CoD: Code on Demand
- MA: Mobile Agents
- State*: new state

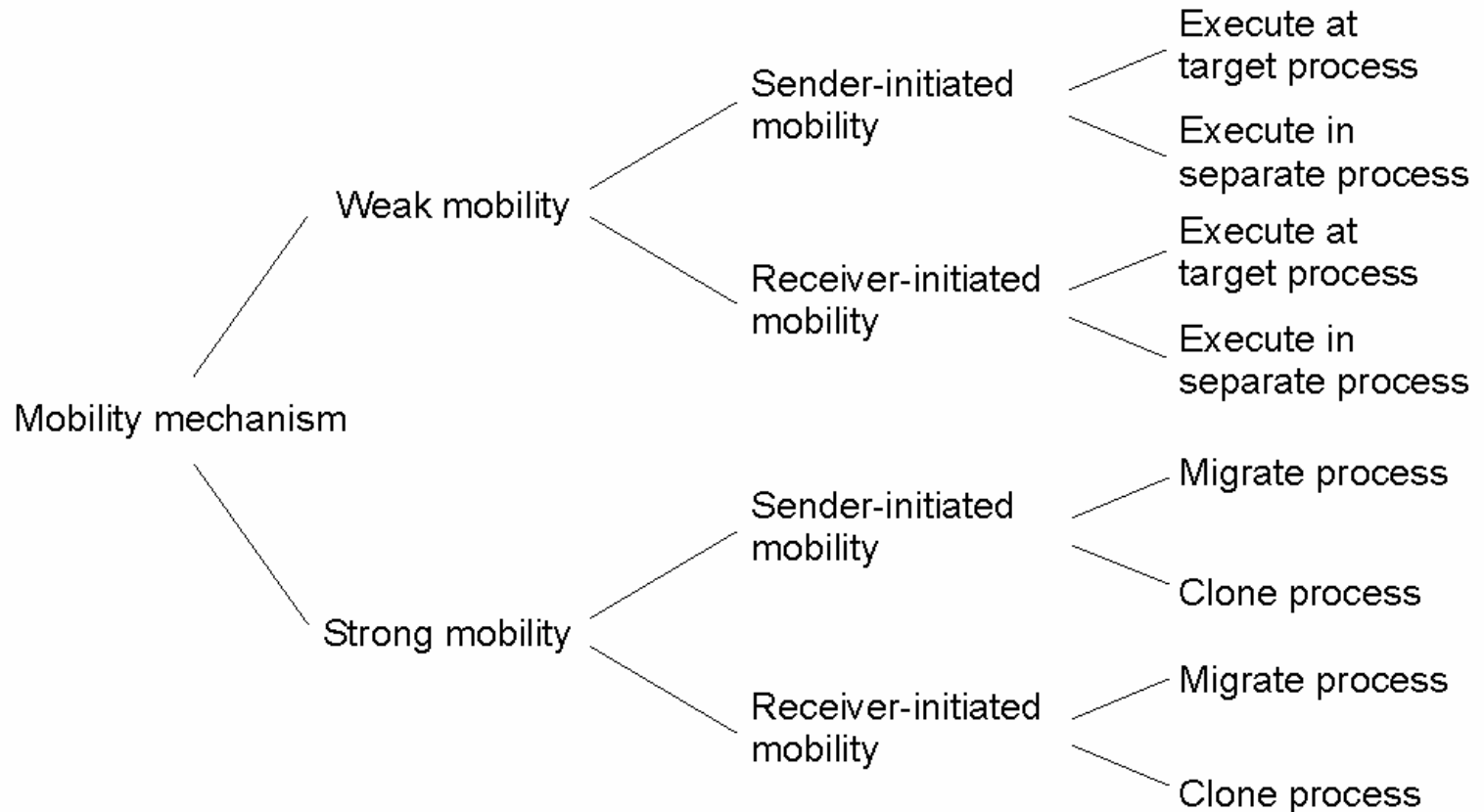


Terminology

- Weak mobility
 - just code & data – only start from initial state
 - e.g. JAVA applets
- Strong mobility
 - also exchange execution segment
- Sender or receiver initiated (push or pull)
- Cloning
 - make exact copy – similar to UNIX *fork()*, but then child process is remote

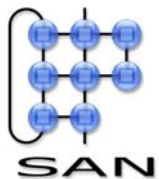


Taxonomy for Code Migration



Dangling references

- **Question:** what to do with *references* to (other) *resources* upon migration
 - depends on “stickyness” of resource to its host
 - fixed (printer), fastened (database) or unattached (file)
 - depends on “uniqueness” of binding
 - precise (URL), value (library function), type (printer)



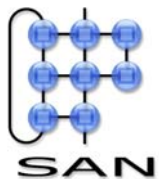
Migration and Local Resources

- MV: move
- GR: Generate Globally Unique Reference
- CP: Copy value
- RB: re-bind to locally available

Resource-to machine binding

**Process-to-
resource
binding**

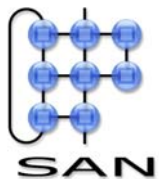
	Unattached	Fastened	Fixed
By identifier	MV (or GR)	GR (or MV)	GR
By value	CP (or MV, GR)	GR (or CP)	GR
By type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)



Migration in Heterogeneous Systems

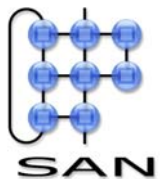
- Problems
 - different Instruction Set Architecture (ISA)
 - different notion of executable, process, thread (OS)
 - different representations of data

- Solutions
 - Define virtual machine
 - language interpretation
 - Limit migration points
 - translate into migratable format



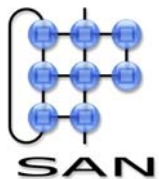
Processes (in distributed context)

- Threads & processes
- Client & Server architecture
- Migration
- Agents
- Example: load balancing



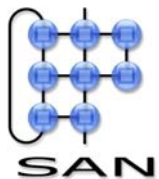
Agent def.: Computing dictionary

- In the client-server model, the part of the system that performs information preparation and exchange *on behalf of* a client or server.
- Especially in the phrase "intelligent agent" it implies *some kind of automatic process* which can *communicate* with other agents to perform some *collective task on behalf of* one or more humans.



Agent def.: Learnthat

- Intelligent Bots. Intelligent Agents are "bots" designed to work for the user in terms of executing transactions (typically search queries at this point) across the web.
- It is assumed that as Intelligent Agents become more sophisticated, they can accomplish more complex tasks for their users. Once they understand their users' requirements, they can act somewhat independently of user interaction, in terms of making online purchasing decisions and other tasks that currently require significant search and user decision making processes.
- Intelligent Agents will also become more useful as industry standards are established.

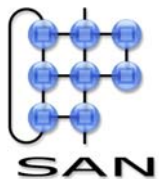


Jennings & Woodridge

- Agent: autonomous process capable of
 - reaction to
 - initiating

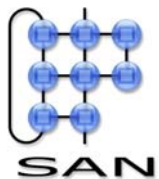
changes in their environment, possibly in collaboration with user, other agents.

- I add: agents are further characterized by
 - goal oriented-ness
 - learning capability



Some goals and classifications

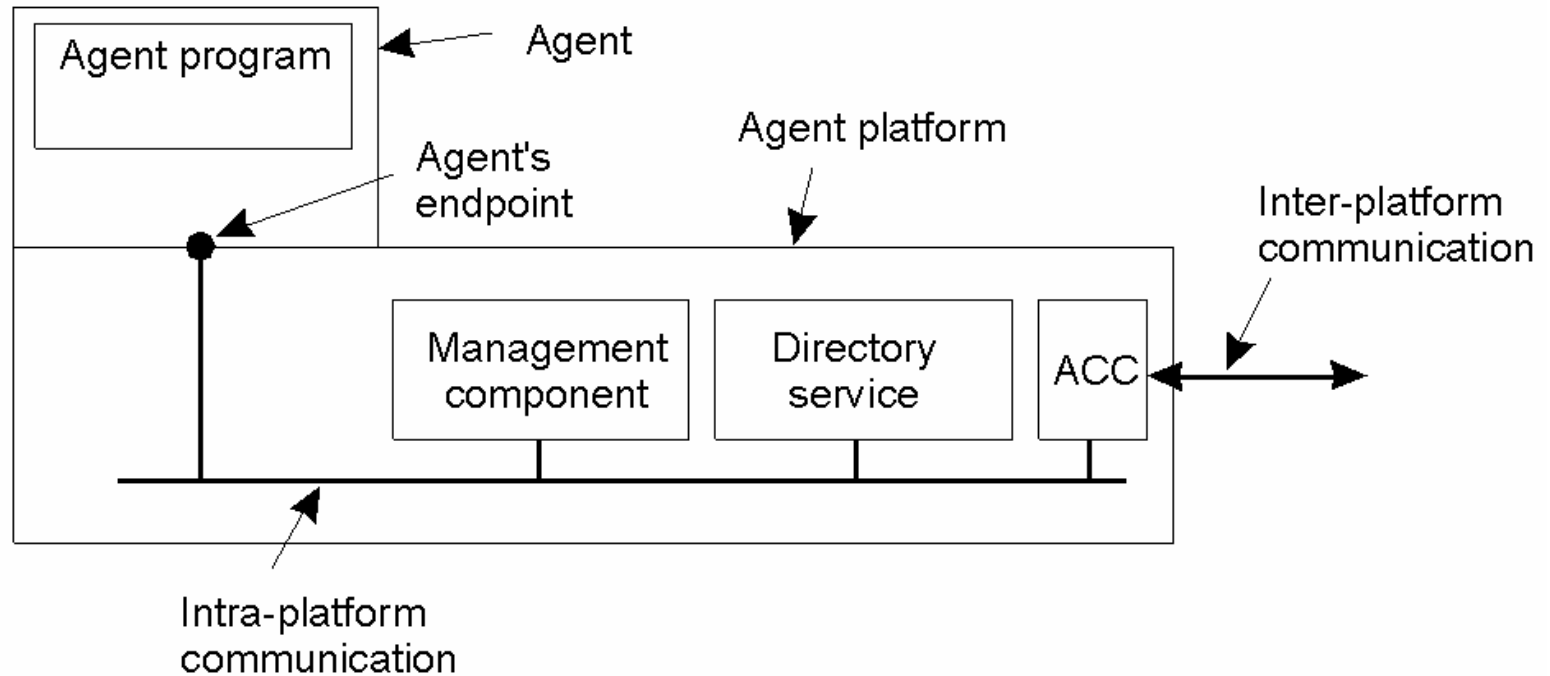
- Collaborative agents (multi-agent systems)
- Mobile agents
- Interface agents (...wizzard...)
 - assist user in using applications
- Information agents
 - collect relevant information



Agent properties

Property	Common to all agents?	Description
Autonomous	Yes	Can act on its own
Reactive	Yes	Responds timely to changes in its environment
Proactive	Yes	Initiates actions that affects its environment
Communicative	Yes	Can exchange information with users and other agents
Continuous	No	Has a relatively long lifespan
Mobile	No	Can migrate from one site to another
Adaptive	No	Capable of learning

Agent Technology



The general model of an agent platform

Agent Communication Languages

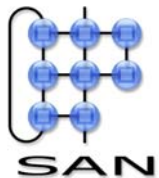
Message purpose	Description	Message Content
INFORM	Inform that a given proposition is true	Proposition
QUERY-IF	Query whether a given proposition is true	Proposition
QUERY-REF	Query for a give object	Expression
CFP	Ask for a proposal	Proposal specifics
PROPOSE	Provide a proposal	Proposal
ACCEPT-PROPOSAL	Tell that a given proposal is accepted	Proposal ID
REJECT-PROPOSAL	Tell that a given proposal is rejected	Proposal ID
REQUEST	Request that an action be performed	Action specification
SUBSCRIBE	Subscribe to an information source	Reference to source

Examples of different message types in the FIPA ACL @ 2002

Example message

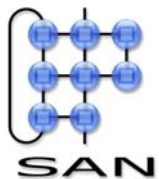
Field	Value
Purpose	INFORM
Sender	max@http://fanclub-beatrix.royalty-spotters.nl:7239
Receiver	elke@iiop://royalty-watcher.uk:5623
Language	Prolog
Ontology	genealogy
Content	female(beatrix),parent(beatrix,juliana,bernhard)

A simple example of a FIPA ACL message sent between two agents using Prolog to express genealogy information.



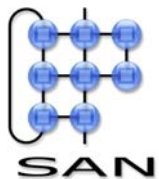
Processes (in distributed context)

- Threads & processes
- Client & Server architecture
- Migration
- Agents
- Example: load balancing



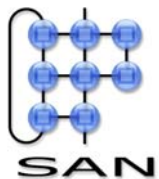
Special case: Load balancing

- Global (DOS) scheduling for optimal resource use
 - system: maximize throughput
 - user: minimize response time
- Dynamic
 - react to dynamically changing load
- Pre-emptive
 - stop process for migration
- No a priori knowledge
 - burdens user with additional specification
- Efficient
 - quick decision
 - precision; real-time
 - balance migration & information gathering

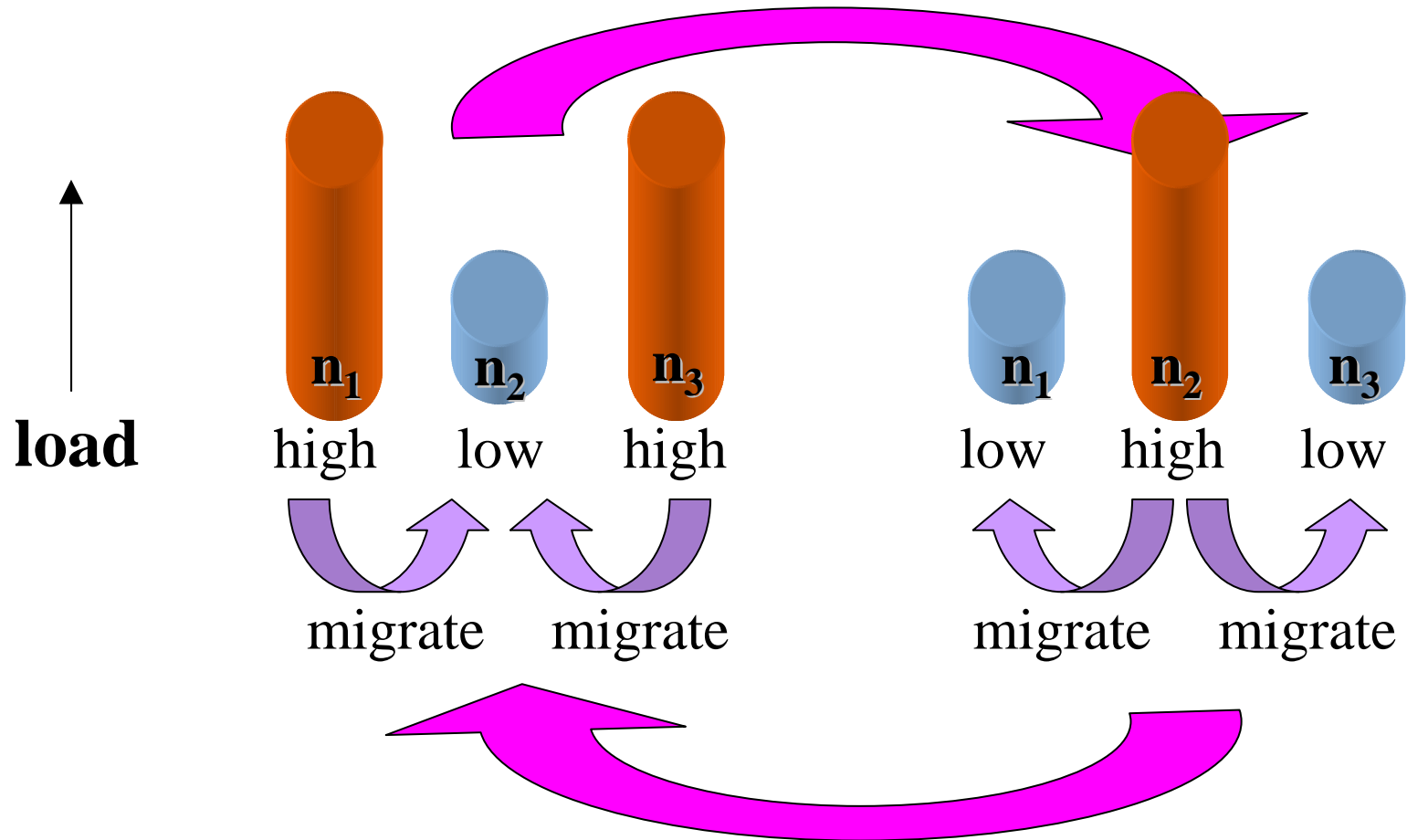


Load balancing stability

- **Definition:** *A load balancing algorithm is **stable** if it is guaranteed to reach a state in which no more processes are migrated, provided that the total (global) load does not change.*

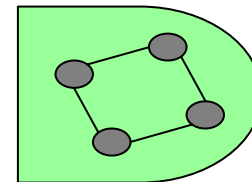
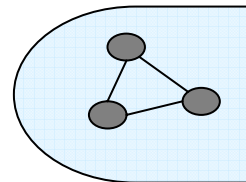
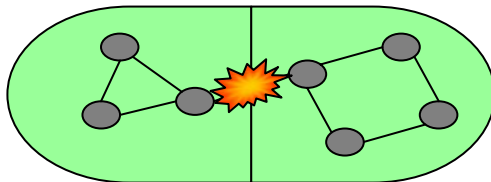


Instability...

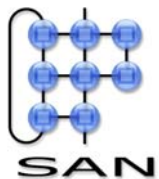
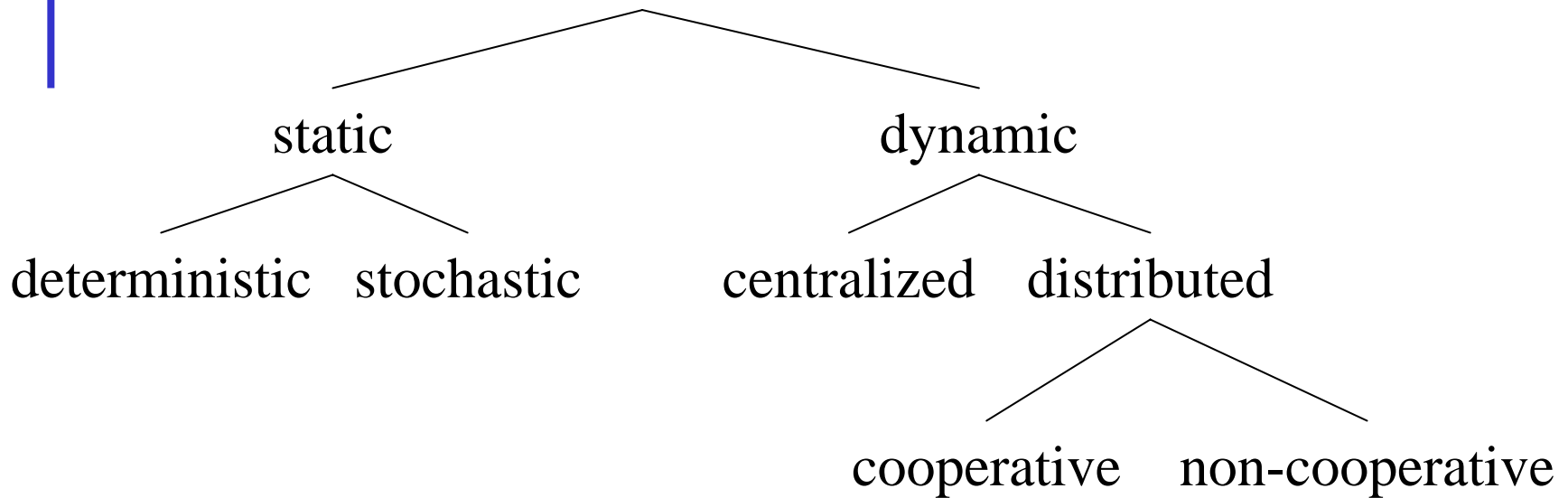


Scalability and fault tolerance

- Scalability of load-balancing
 - no significant effect of scaling the network
 - limit information gathering, e.g., probe m out of n machines
- Fault tolerance
 - node failure: continue for remaining nodes
 - link failure: continue for connected nodes



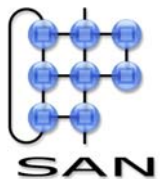
Taxonomy of Load Balancing Algorithms



Dynamic distributed load-balancing

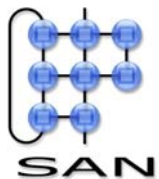
- Load estimation policy
- Process transfer policy
- Policy for exchange of load information
- Priority assignment policy
- Policy for limiting migration

- Policy: 'rule', 'method'



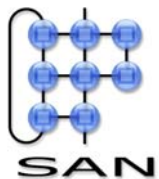
Load estimation

- Useful parameters are, for example:
 - # processes on a node (at time of estimation)
 - total resource demands of these processes
 - instruction mixes of processes
 - architecture and speed of resource
 - remaining service time
 - fixed: then method = #processes
 - unknown but varying: time needed is equal to time used
 - distribution: assumes a distribution of service times
 - CPU utilization = #CPU cycles/unit of time
 - observe regularly



Process transfer policy

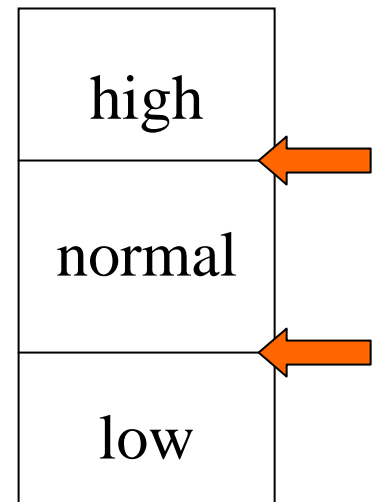
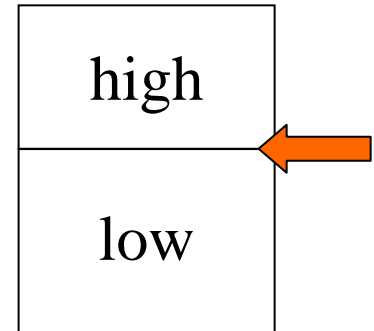
- Idea: transfer processes from heavily loaded nodes to lightly loaded nodes
- Threshold: criterion to decide whether load is high or low
 - dynamic: depends on current total load
 - threshold: $aw^* rpp_j$
 - aw = average workload of 'all' nodes (computed regularly)
 - rpp_j = relative processing power of node j
 - static: fixed, related to capacity
- Source: try to off-load process if load is high
- Destination: accept (ask for) new process if load is low



Single & double threshold

- Single threshold: may be unstable
 - off-load waiting process to idle processor
- Double Threshold (high-low policy)
 - use 'high' and 'low' mark

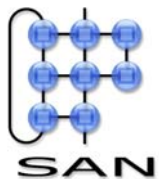
	dispatch	accept
high	yes	no
normal	no	no
low	no	yes



Process Transfer Initiative

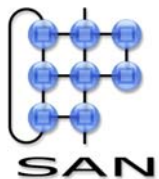
- Sender initiated
 - heavy loaded nodes seek to off-load work
 - may work without pre-emptive migration

- Receiver initiated
 - lightly loaded nodes look for load
 - typically requires pre-emptive migration
 - more expensive (need to record execution state)



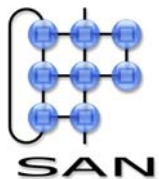
Location policy: where to off-load

- Random
 - probe random destinations until threshold allows transfer
- Minimum load
 - probe load at N random destinations
 - transfer process to node with lowest load that is below threshold; execute locally otherwise
- Bidding
 - each node is seller (manager) and buyer (contractor)
 - seller broadcasts a request for bids
 - buyers send their bids containing info concerning node capabilities; e.g. available processor/memory capacity
 - manager approaches bidders (from best to worst)



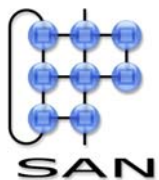
State information exchange policy

- Needed to balance load but consumes resources
- Polling
 - active checking
- Periodic Broadcast
 - heavy network traffic
 - poor scalability (#messages increases)
 - data may not be needed
- Improvements
 - send data only after (significant) load-change
 - send data after request (by a high-load node)
 - reply only if can help in balancing



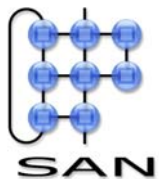
Process priority policy

- Nodes have local and remote processes. Which has priority?
- Selfish (best local response)
 - local > remote
- Altruistic (best global response)
 - remote > local
- Intermediate
 - if #local processes > #remote processes
then local > remote
else remote > local



Policy for Limiting Migration

- Uncontrolled
 - may lead to instability
- Controlled
 - set a maximum to number of migrations per process (process keeps track of migration count)
 - 1: irrevocable (typically static)
 - k : may be decided (dynamically) based on process characteristics



Open ends

- Process selection policy: which process to off-load?
- Compare time to complete on local node to time to complete at remote node.
- Take transfer time (migration delay) into account
 - distance between nodes
 - size of process

