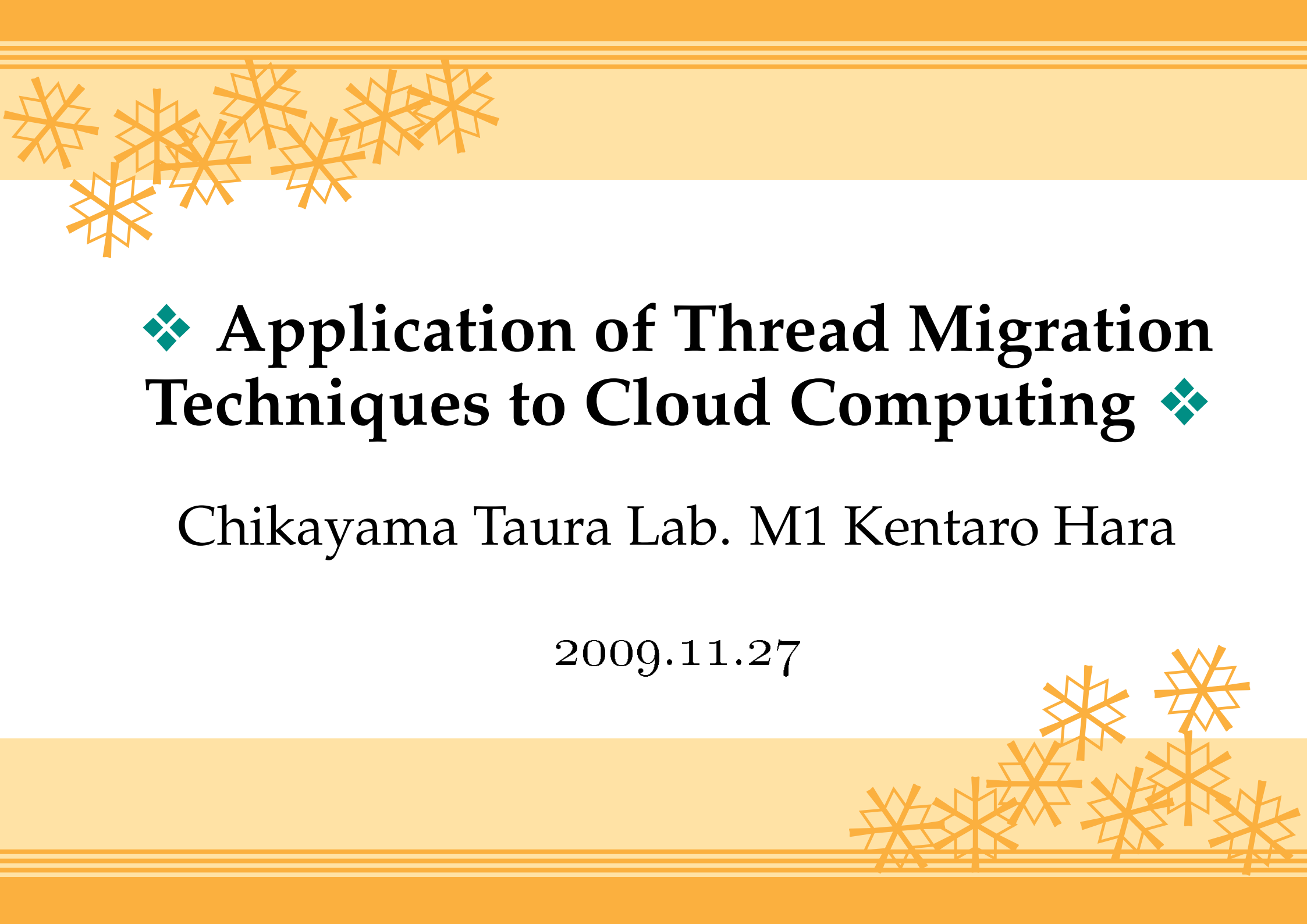


❖ Application of Thread Migration Techniques to Cloud Computing ❖

Chikayama Taura Lab. M1 Kentaro Hara

2009.11.27

Agenda

- What is Cloud Computing?
- Cloud Computing Services
  - Amazon EC2
  - Google App Engine
  - Thread migration-based model
- Kernel Thread Migration
  - Iso-address
- Fast Memory Migration
  - Pre-copy
  - Post-copy
- Conclusions

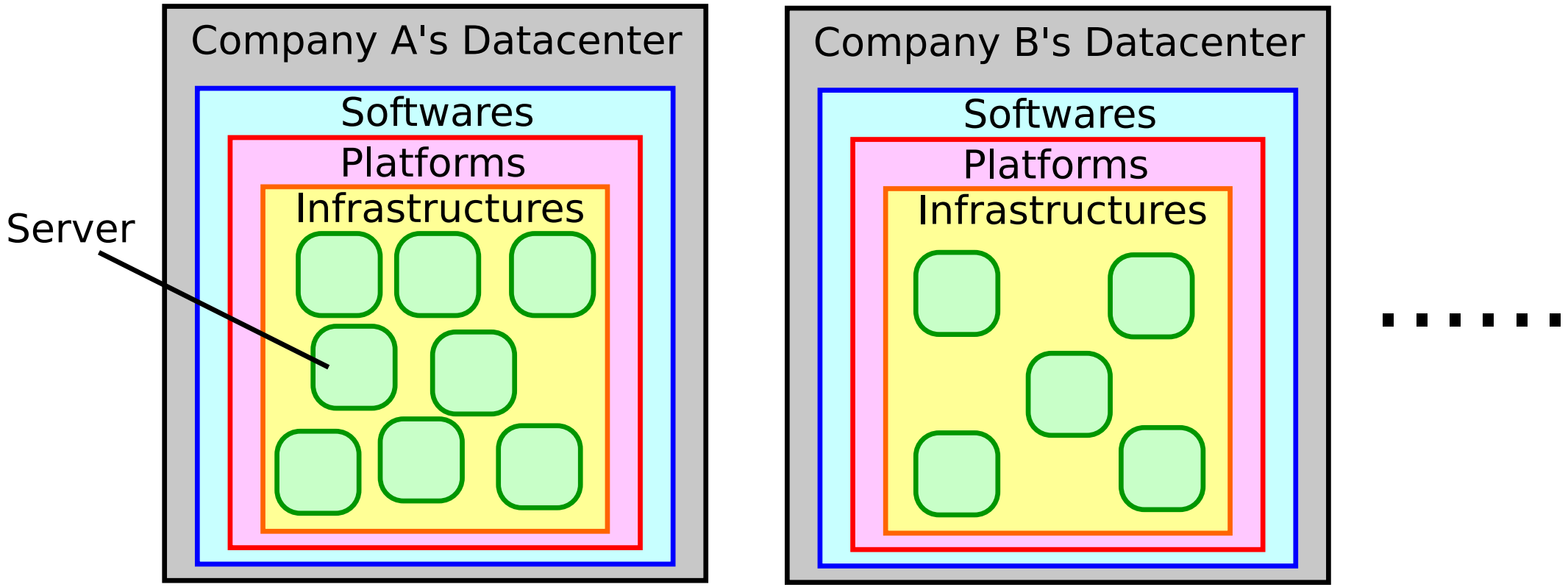
1. What is Cloud Computing?

Backgrounds

- More and more apps require many computational resources
  - → Web apps
    - ♦ SNS
    - ♦ Online game
  - → High performance computing apps
    - ♦ DNA analysis
    - ♦ Earthquake simulation
  - → etc...

A conventional approach : A private datacenter

- Build up a private datacenter, and run apps there
  - → Datacenter = Infrastructures + Platforms + Softwares

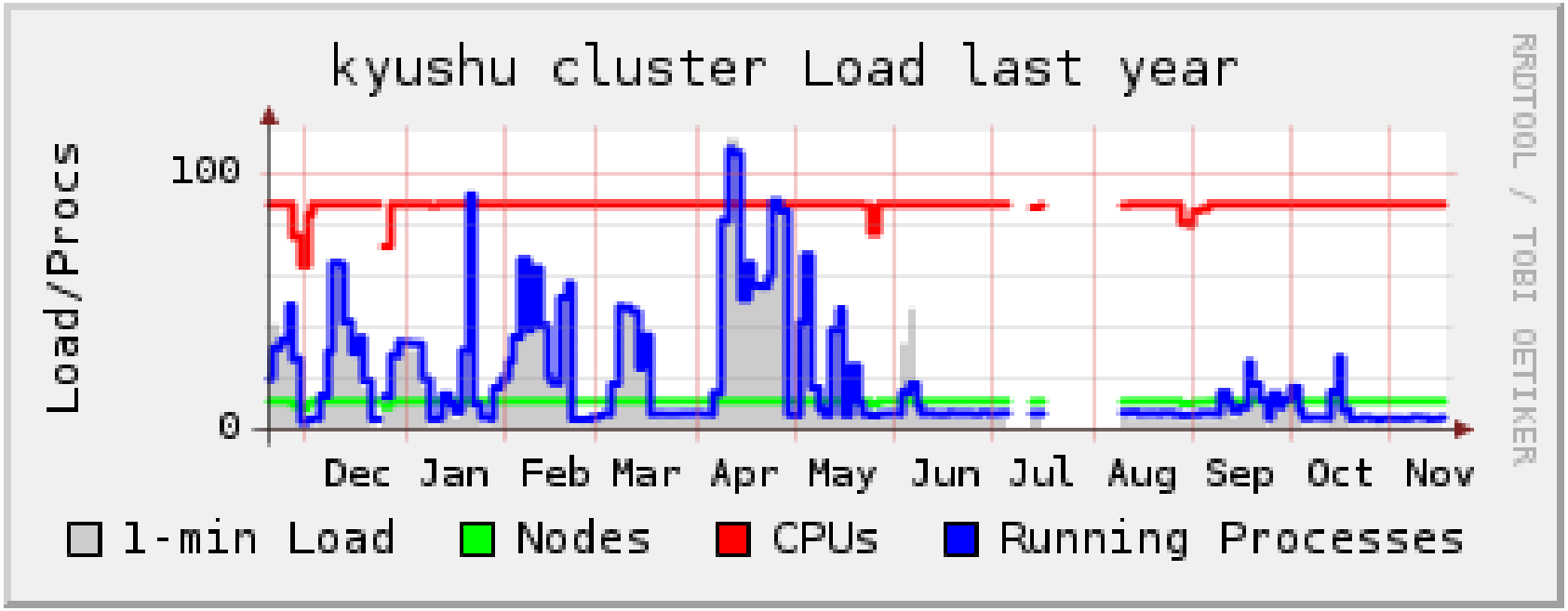


Demerits of a private datacenter(1)

- High management cost
  - → The purpose is NOT to manage a datacenter
  - → The purpose is to run apps
- Difficulties in estimating the adequate number of servers
  - → Underestimation leads to high loaded condition...
  - → Overestimation leads to excessive capacity, which means wasted investment...

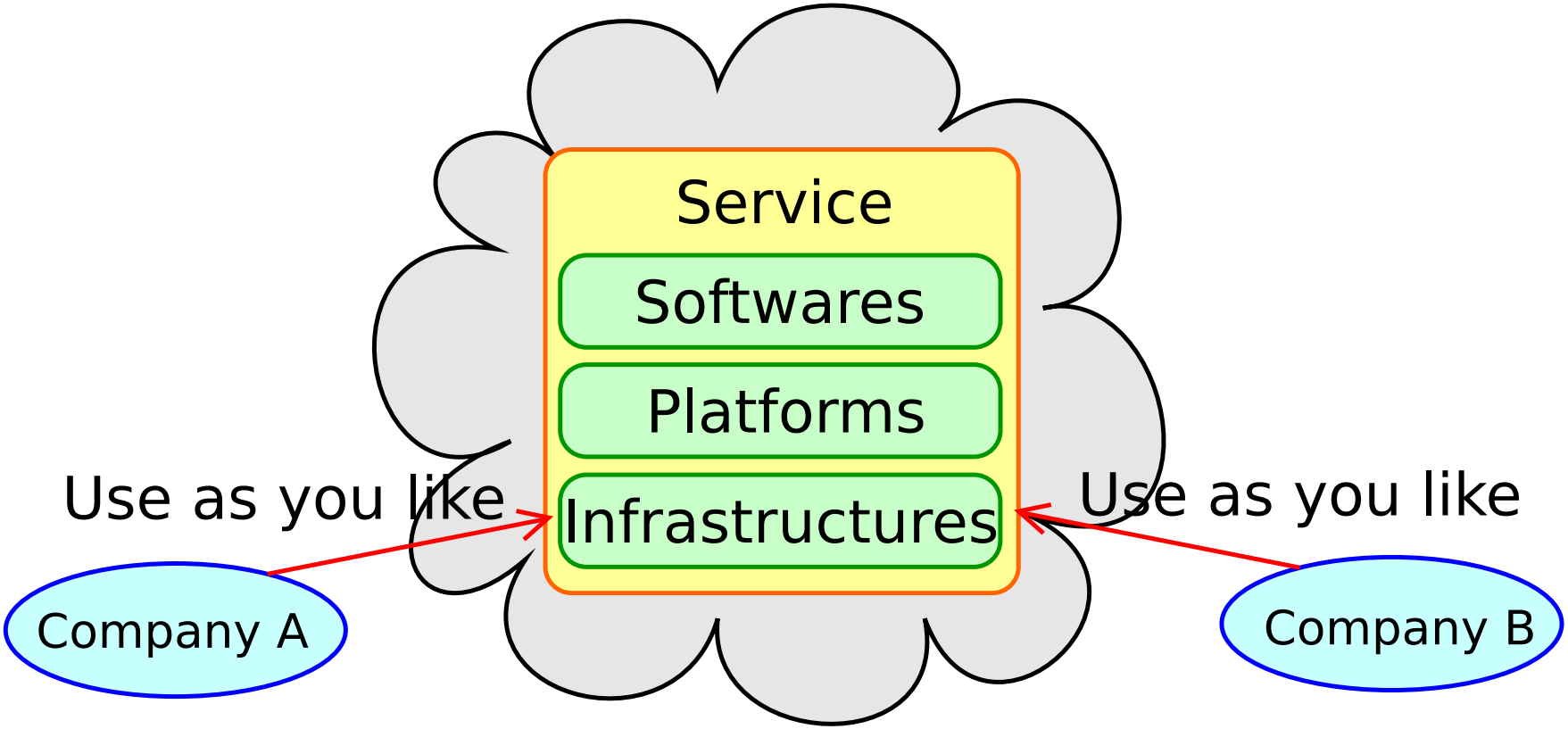
Demerits of a private datacenter(2)

- A datacenter with statically fixed resources cannot adapt to dynamic load fluctuation efficiently
- General trend[Armbrust et al, 2009]
  - → Average server utilization is about 5-20%
  - → The peak workload exceeds the average by factors of 2 to 10



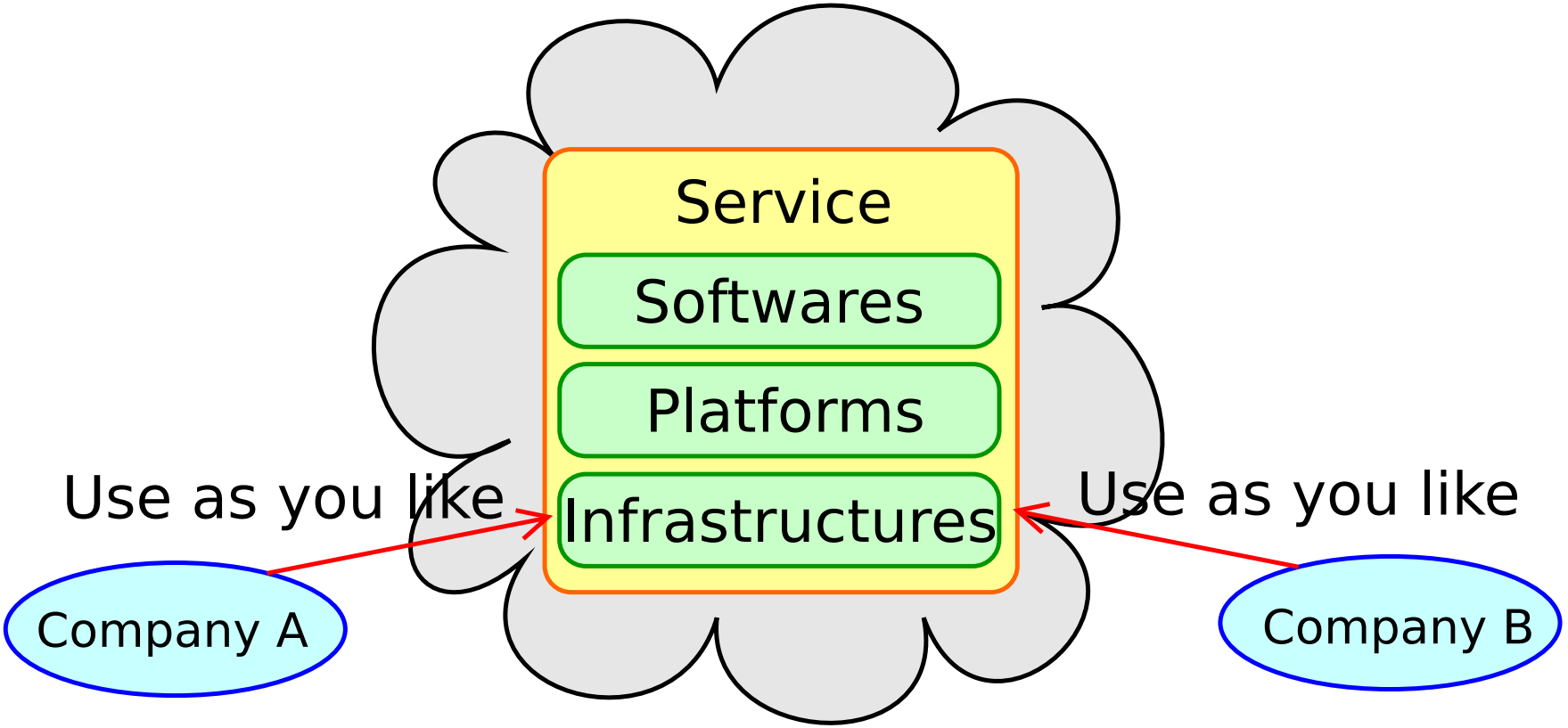
A new approach : Cloud Computing

- A provider provides many resources as a service
- A user can use only as many resources as needed when needed in a pay-as-you-go system



Merits of Cloud Computing

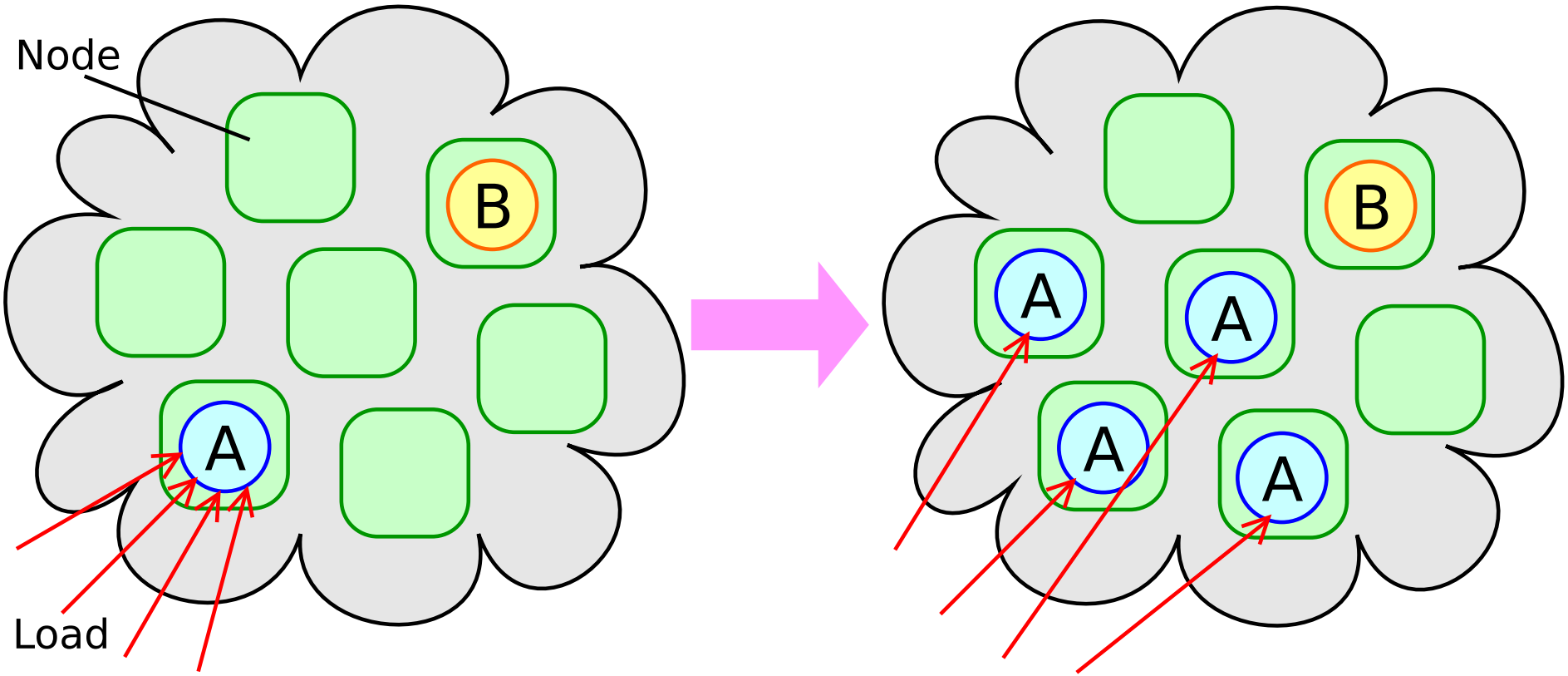
- No specialized knowledge about troublesome server management
- Efficient adaptation to dynamic load fluctuation



An example of Cloud Computing(1)

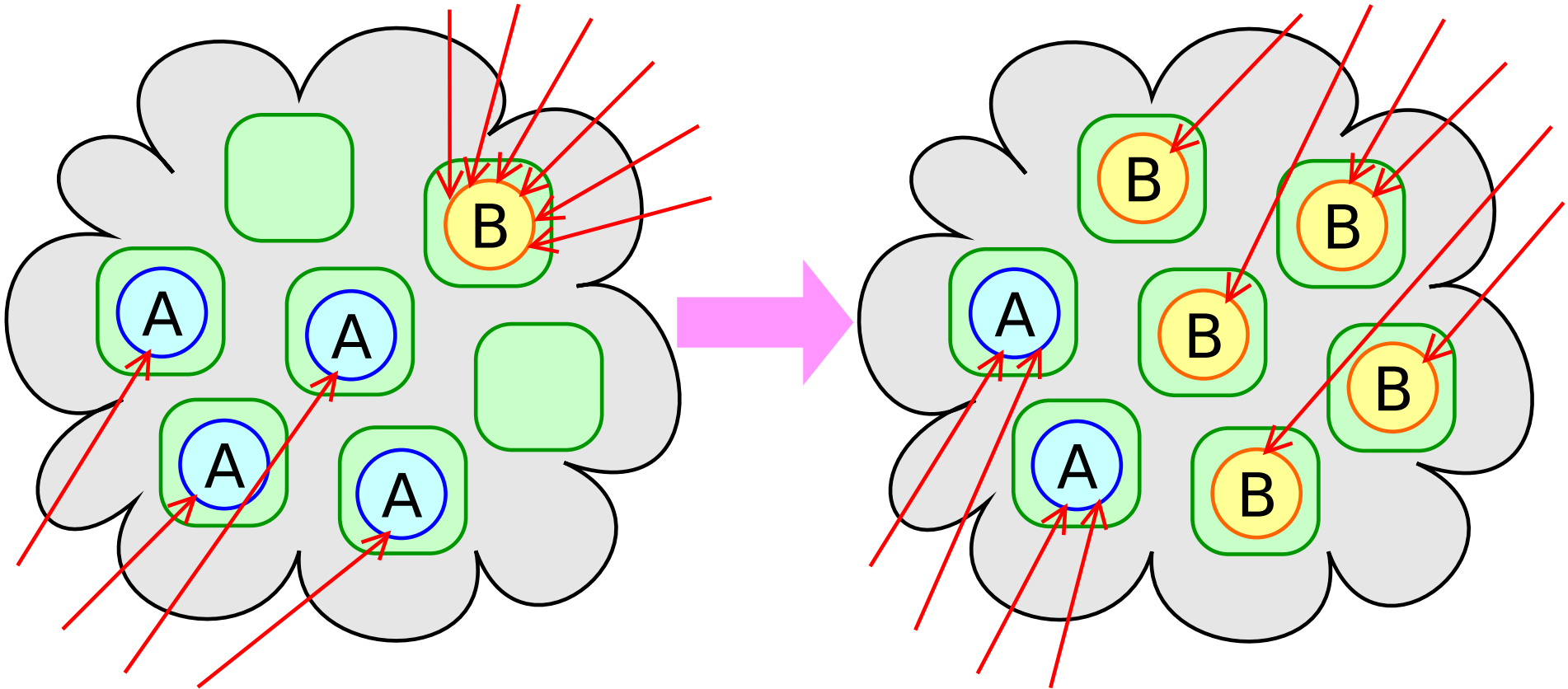
- When user A's load increases, the resources which user A can use increase

  → Load gets balanced



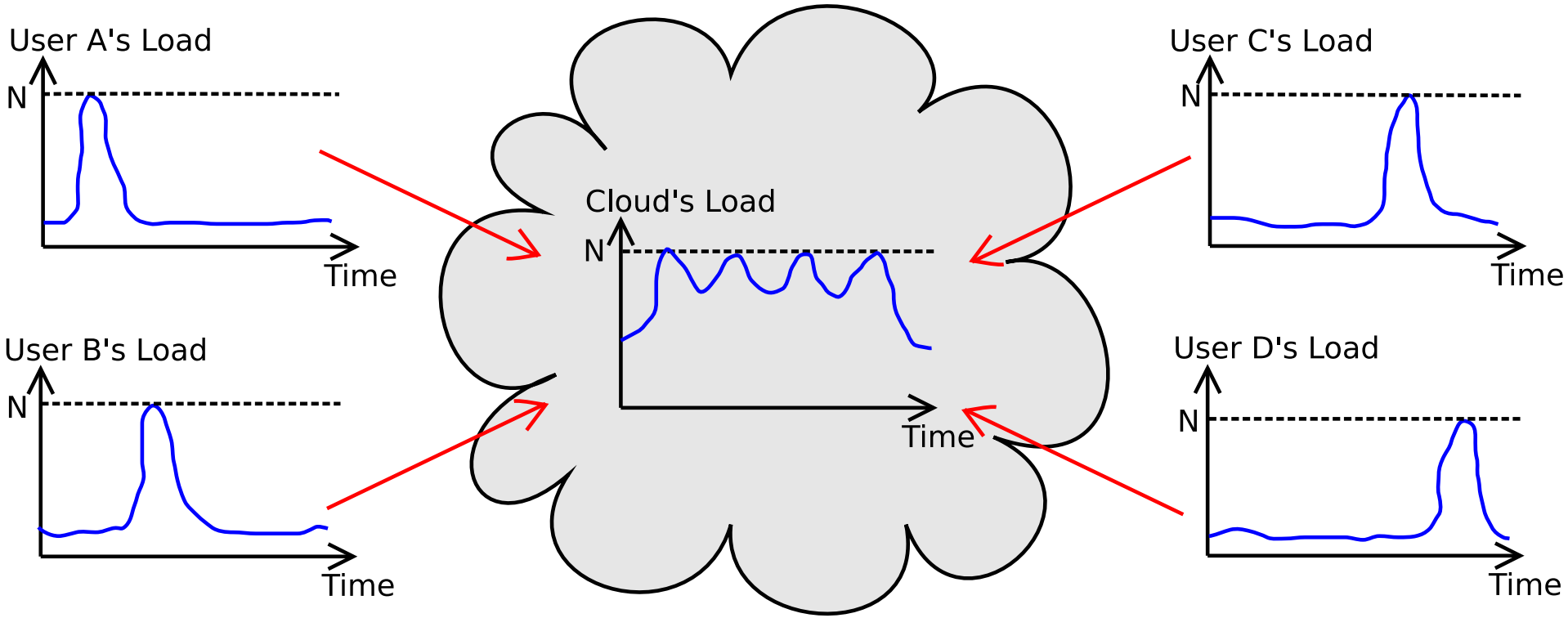
An example of Cloud Computing(2)

- When user B's load exceeds user A's load, the resources which user B can use increase, decreasing the resources which user A can use
  - → Shared resources are scheduled (according to some policies)



The essence of Cloud Computing

- To absorb load fluctuation by sharing many resources with many users[Taura, 2009]
  - "Better to use 10000 servers with 10000 users than to use 10 servers with 10 users"

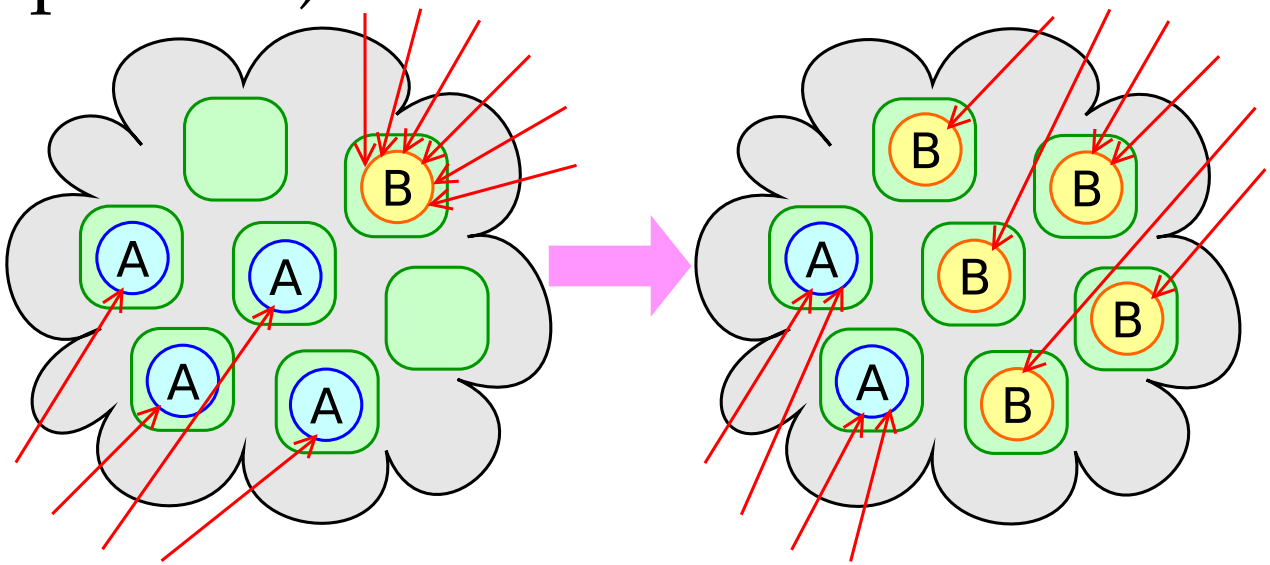


Requirements for Cloud Computing services

- Although Cloud Computing services are featured in many ways, ...
- Common requirements :

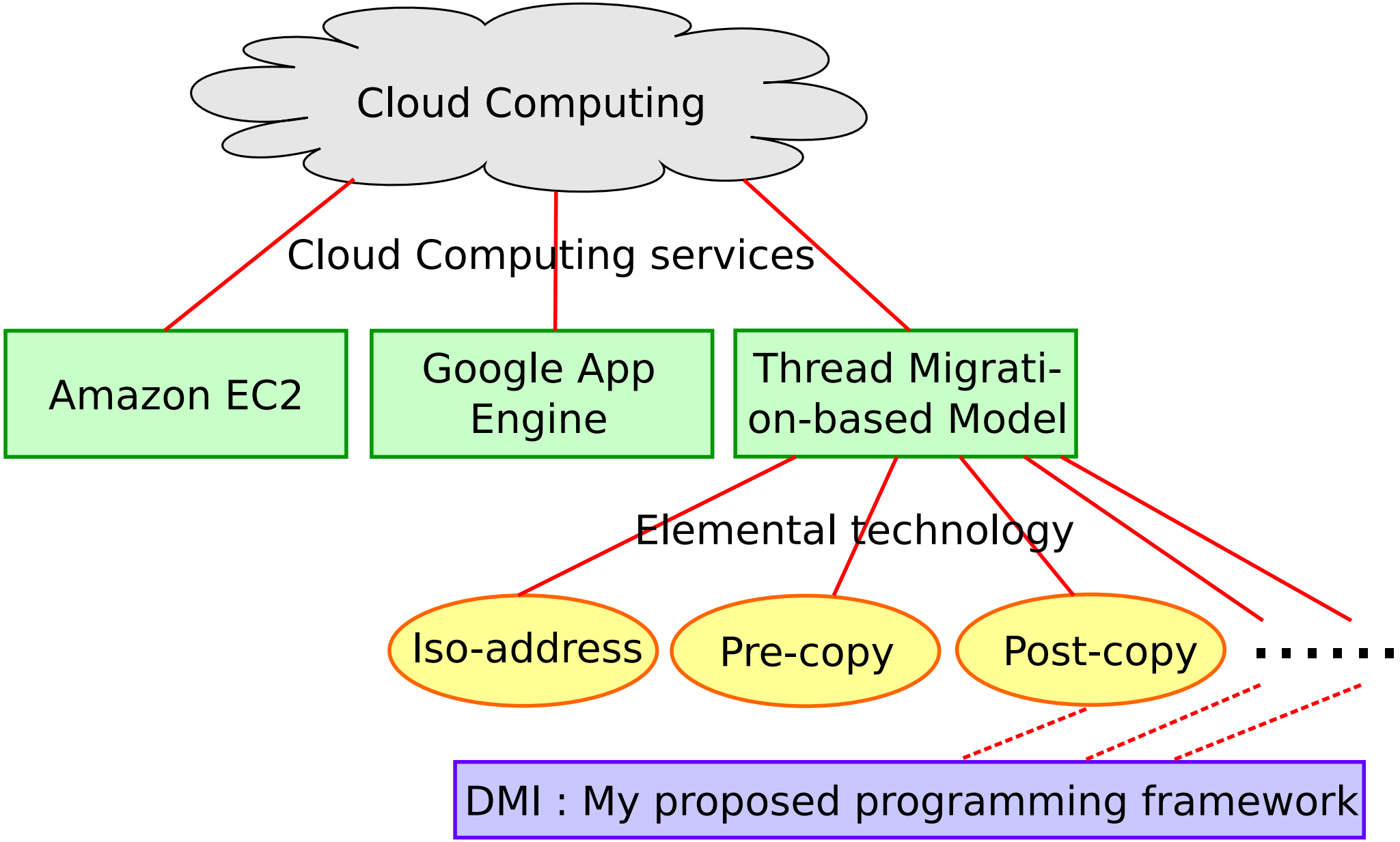
  (1) To support flexible scale-up/scale-down in response to load increase/decrease

  (2) To schedule shared resources between users (according to some policies)



Scale-up/Scale-down + Resource Scheduling

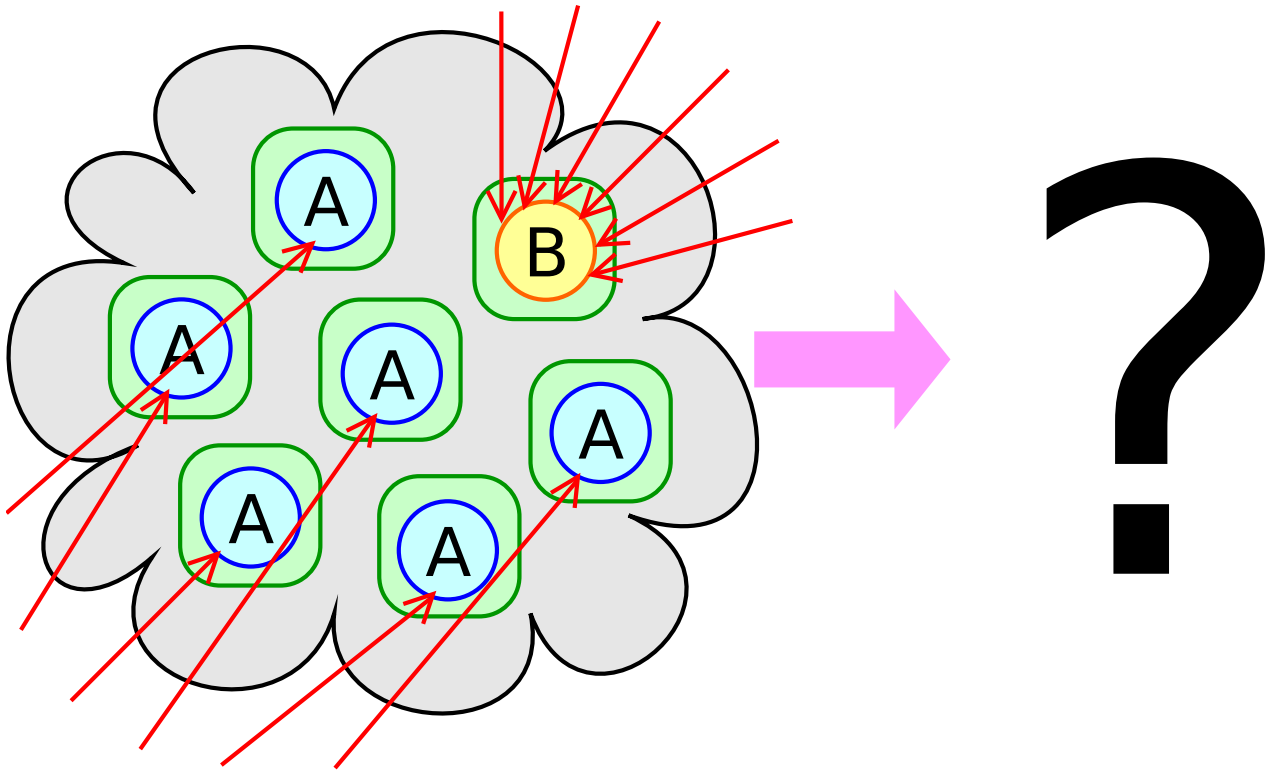
Road map


Cloud Computing
Cloud Computing services
Amazon EC2
Google App Engine
Thread Migrati-on-based Model
Elemental technology
Iso-address
Pre-copy
Post-copy
DMI : My proposed programming framework

2. Cloud Computing Services

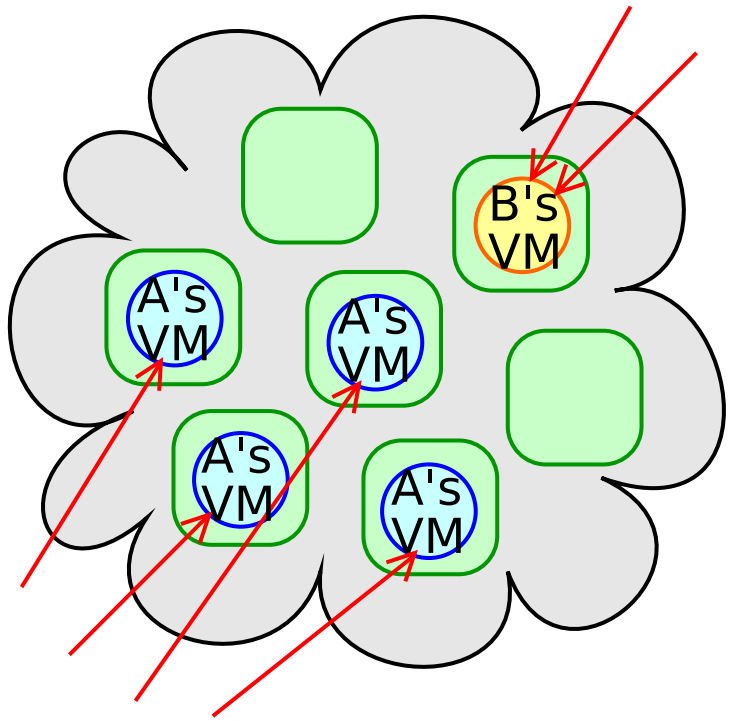
Key points of the following discussion

- What is a unit of scale-up/scale-down in each service?
- How does each service schedule resources?
  - → How does each service handle the following situation?



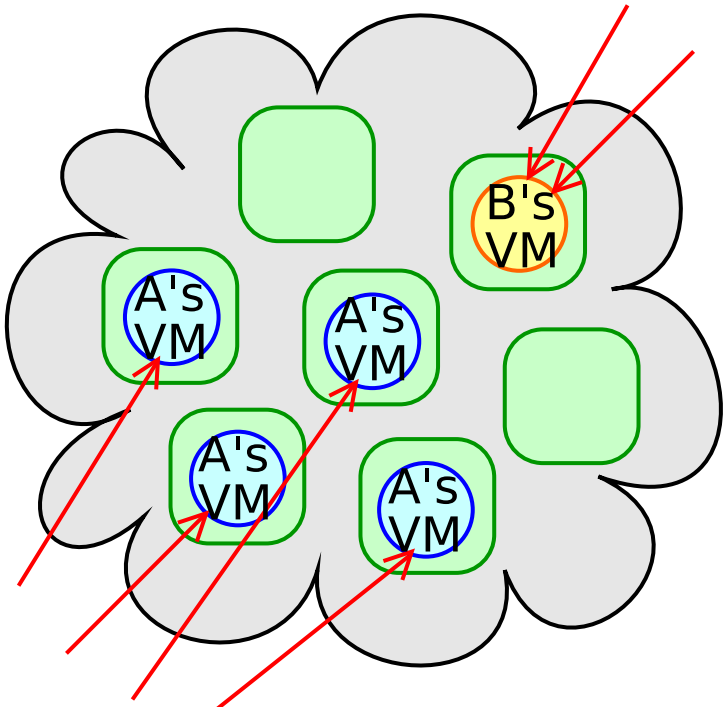
Amazon EC2(1)

- A VM is a unit
- A user can scale up/down apps by starting/stopping VMs when needed
- ◎ A VM provides general computational environment
  - → Possible to run long-time apps



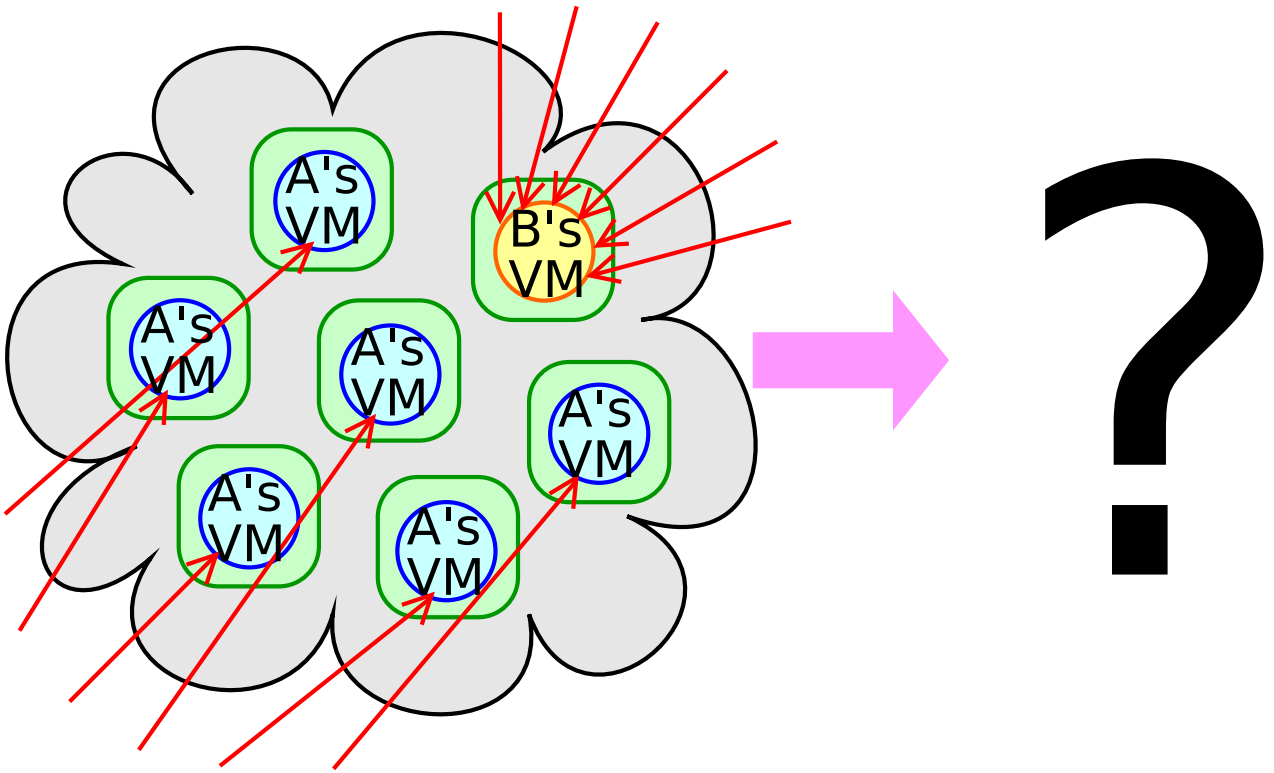
Amazon EC2(2)

- ✕ Charged hourly per VM (not per actually used CPU cycle)
  - → because a VM consumes much resource even if it only stays
- ✕ Slow adaptability to load fluctuation
  - → because starting/stopping a VM takes minutes



How does it schedule resources?(1)

- What happens in the following situation?

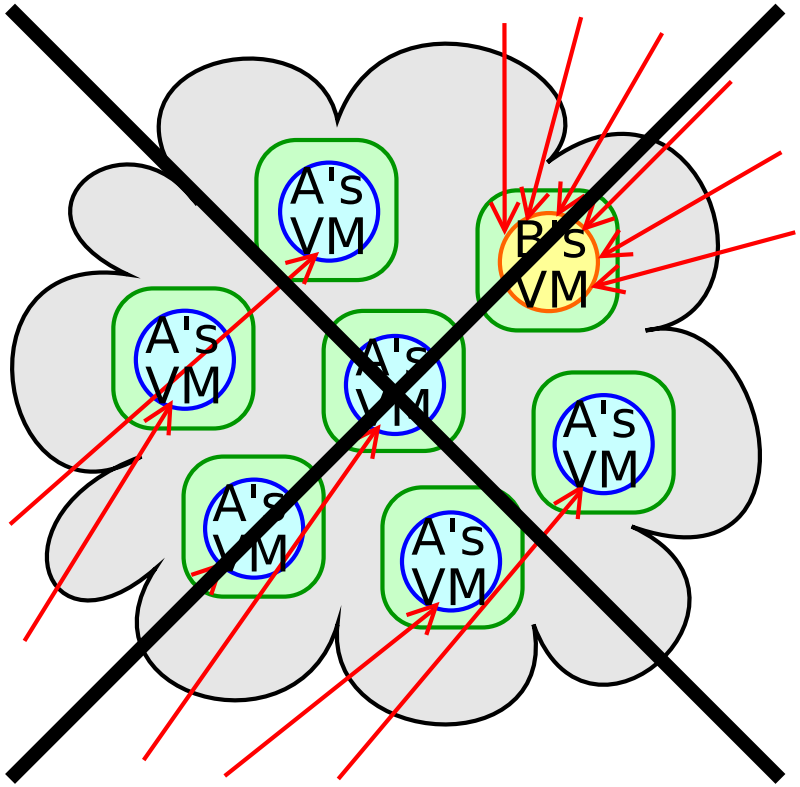


How does it schedule resources?(2)

- Rule : One user can boot only up to 20 VMs
  - → A detailed application is required to boot over 20 VMs
- Thus, temporal surging load does not mean the increase of the overall number of VMs
  - → This enables Amazon EC2's administrators to estimate the overall number of VMs, which enables administrators to manage resources not to run out of them

How does it schedule resources?(3)

- In essence, Amazon EC2 prevents the following situation from happening, at the expense of rapid adaptability to load fluctuation
  - → × Slow adaptability to load increase

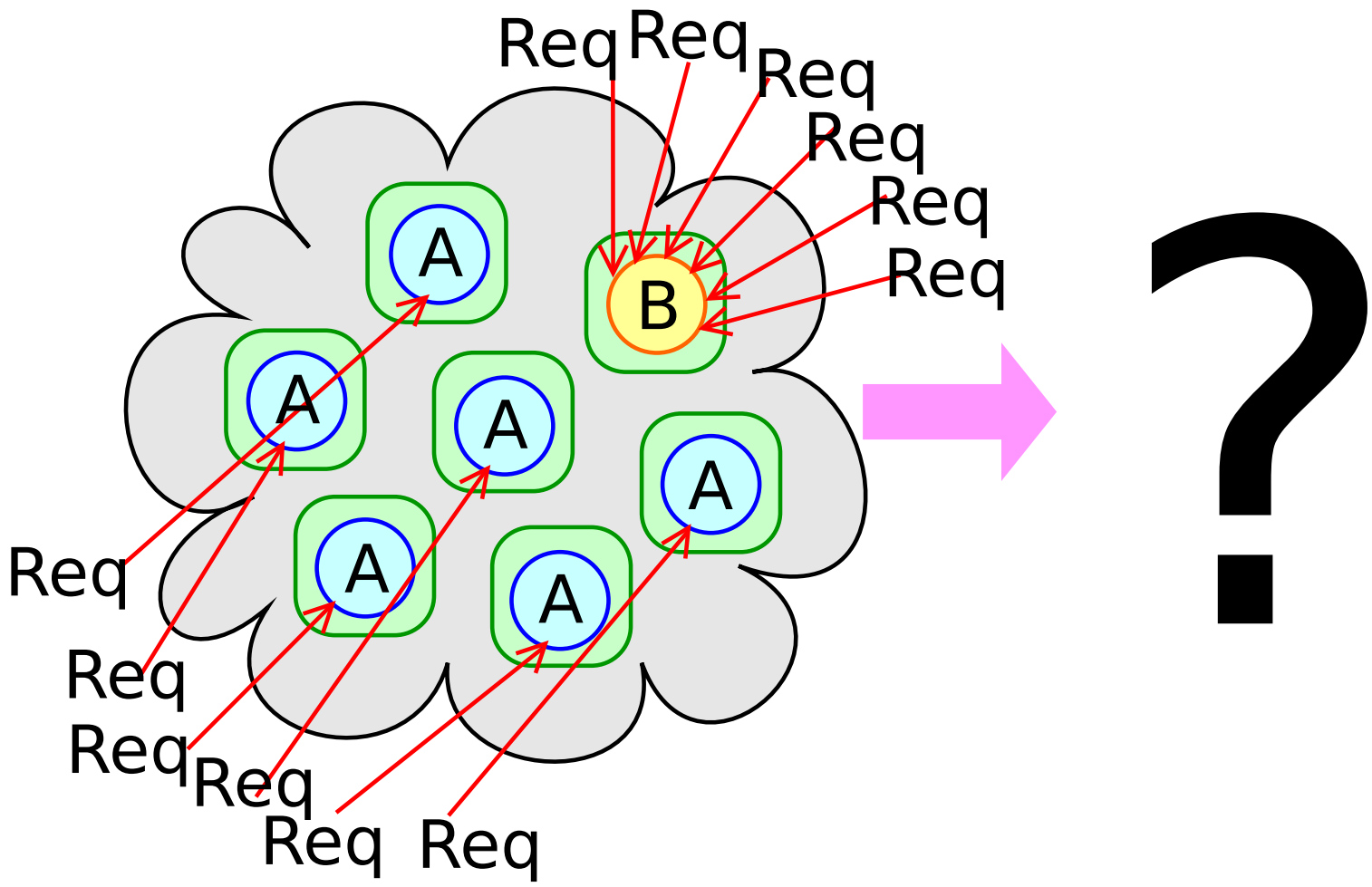


Google App Engine(GAE)

- A user can run Web apps written in Python/Java on the Google's efficient infrastructure
- A request from a Web client is a unit
- ◎ GAE automatically and rapidly scale up/down apps in response to request load fluctuation
  - → GAE can handle up to 7400 requests per minute (even in a free default quota)
- ◎ Charged per CPU cycle, bandwidth, ...
  - → You only pay for what you REALLY use

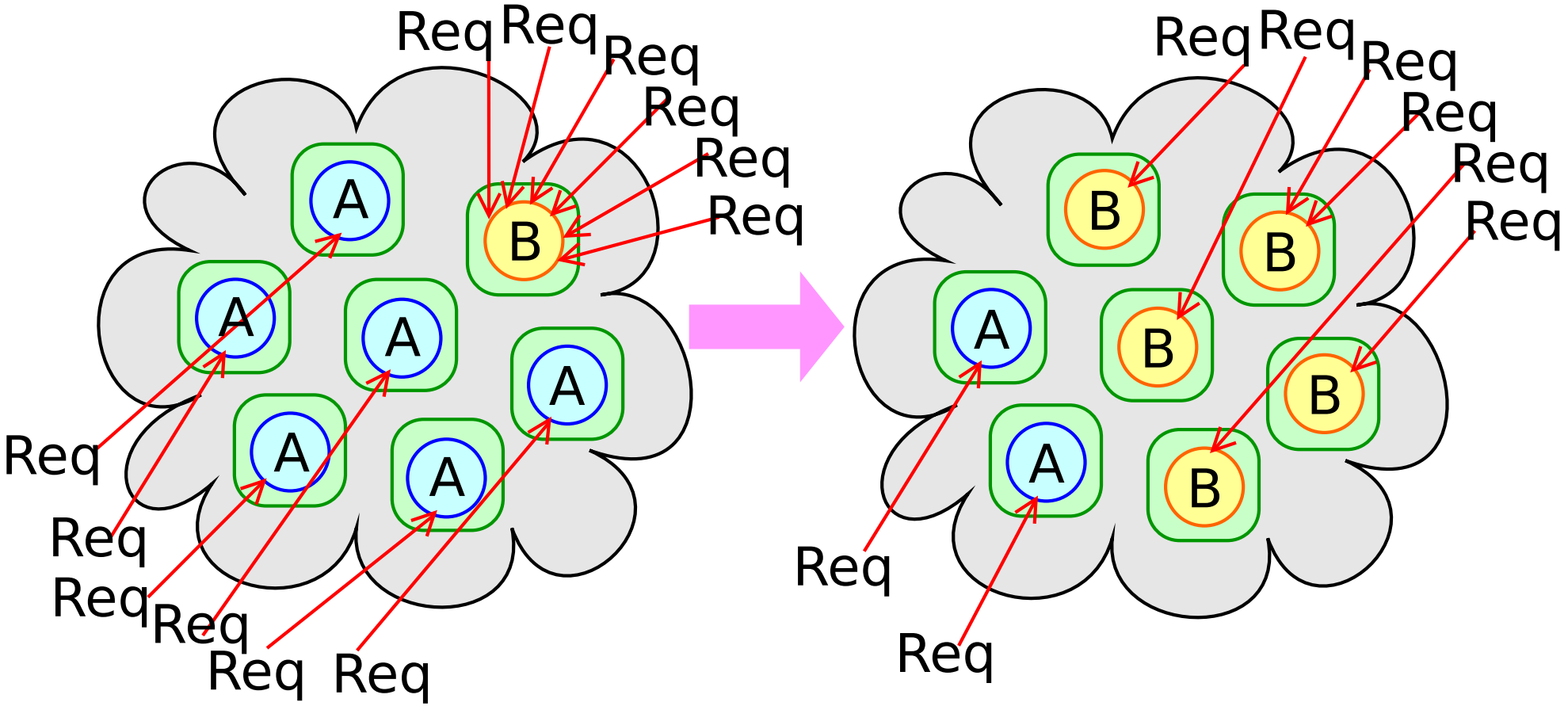
How does it schedule resources?(1)

- What happens in the following situation?



How does it schedule resources?(2)

- Rule : Each request must be processed within 30 seconds
  - → Requests over 30 seconds are killed
- In essence, GAE can schedule resources at short-time intervals by limiting each request's processing time



How does it schedule resources?(3)

- ✕ Thus, GAE supports only short-time apps
  - → Specialized for typical Web apps
  - → Almost impossible to run high performance numerical computing
    - ex : sorting, simultaneous equations solver, ...

A comparison between Amazon EC2 and GAE

	Amazon EC2	GAE
Unit of scale-up/scale-down	VM	Request
Resource consumption	Large	Small
Billing granularity	Coarse	Fine
Adaptability to load fluctuation	Slow	Rapid
Domain of targeted apps	Large	Small
Long-time apps	OK	NG

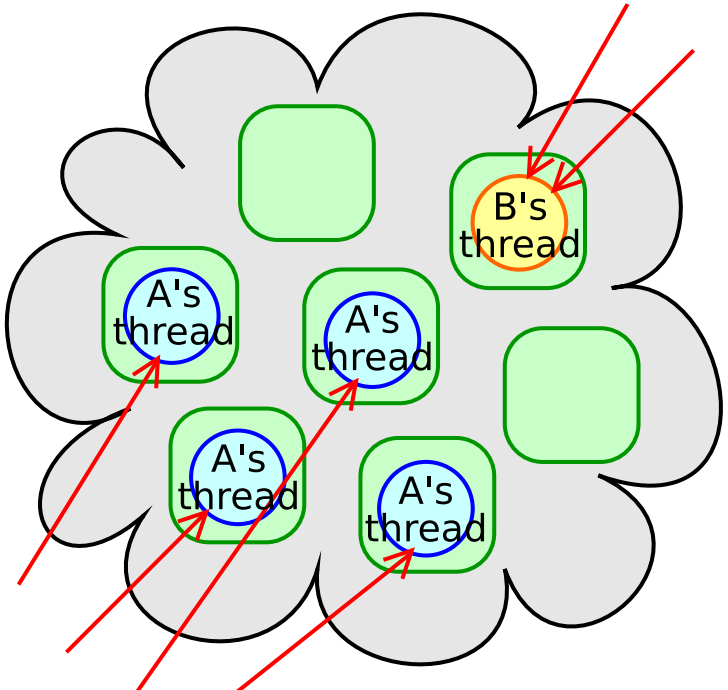
A “middle” approach

- A thread migration-based model

	Amazon EC2	Thread migration-based Model	GAE
Unit of scale-up/scale-down	VM	Thread	Request
Resource consumption	Large	Middle	Small
Billing granularity	Coarse	Middle	Fine
Adaptability to load fluctuation	Slow	Middle	Rapid
Domain of targeted apps	Large	Middle	Small
Long-time apps	OK	OK	NG

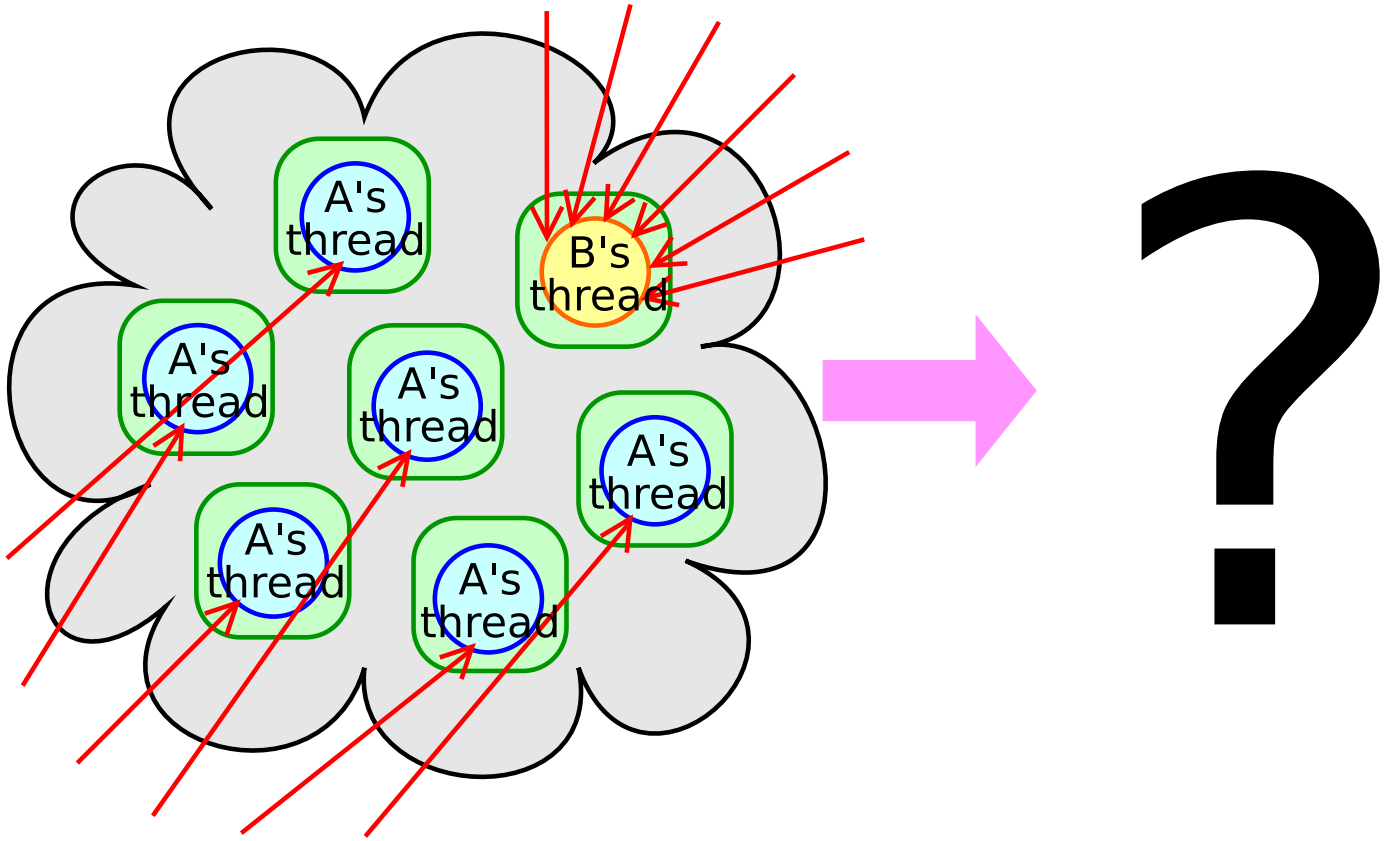
A thread migration-based model

- A thread is a unit
- ◎ Rapid scale-up/scale-down in response to load fluctuation
  - → because a thread is lighter than a VM
- ◎ No running time limit for each thread
  - → Possible to run long-time apps



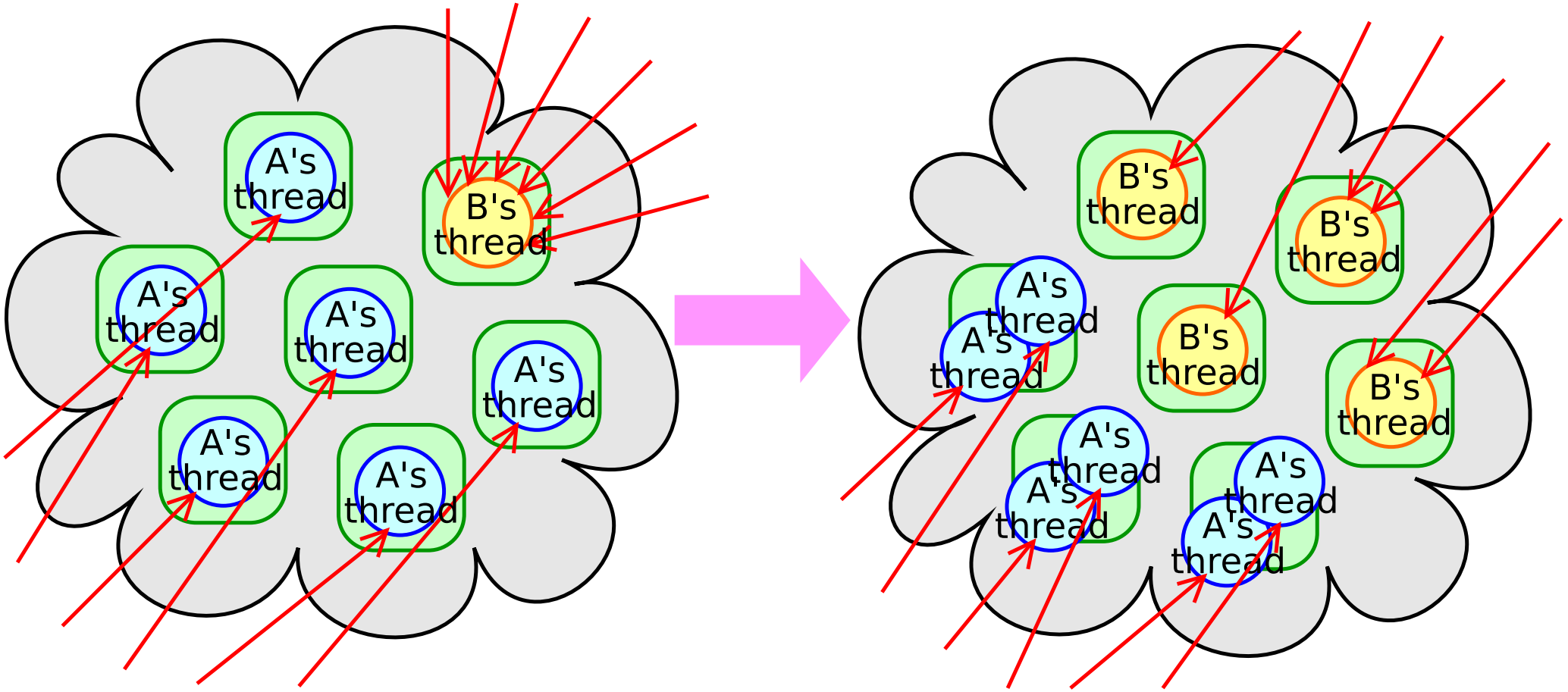
How does it schedule resources?(1)

- What happens in the following situation?



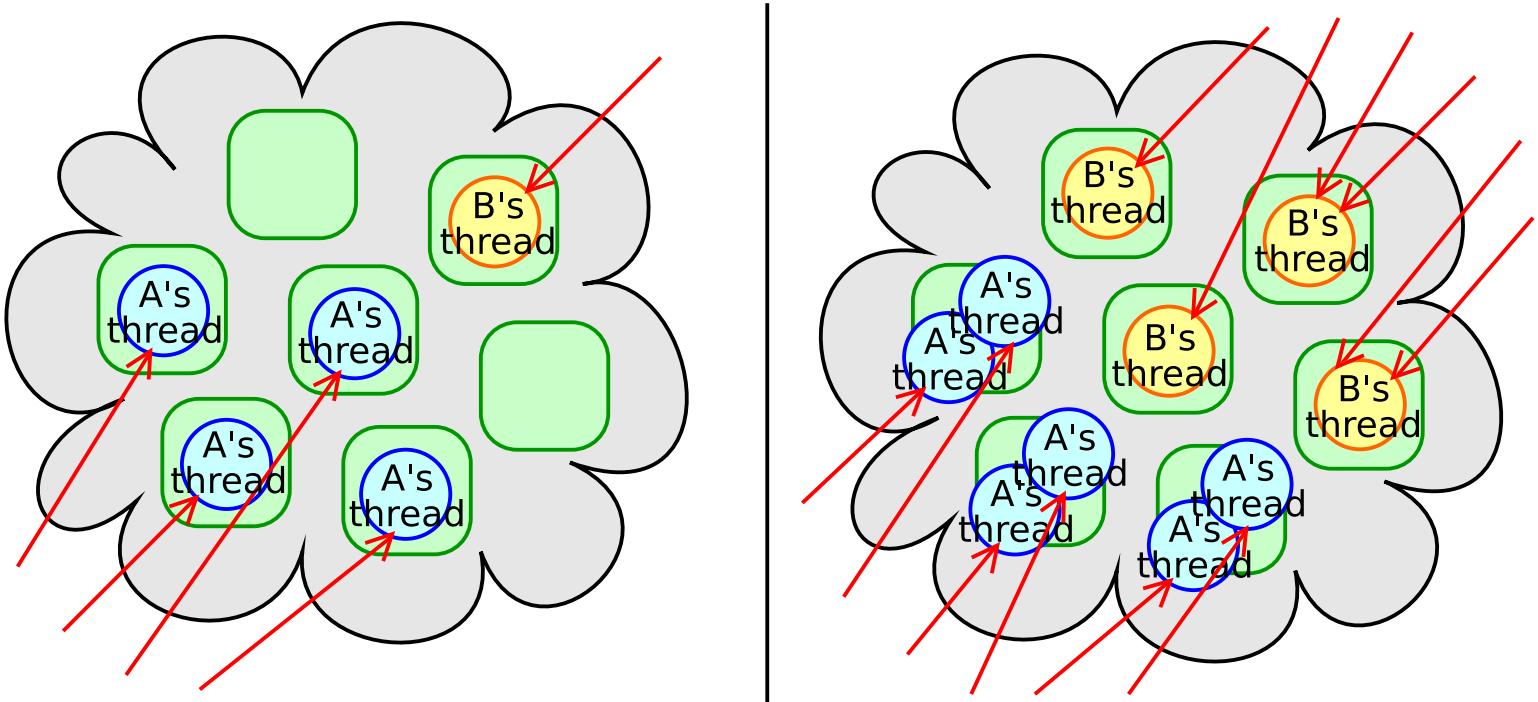
How does it schedule resources?(2)

- Schedule resources by migrating running threads when needed



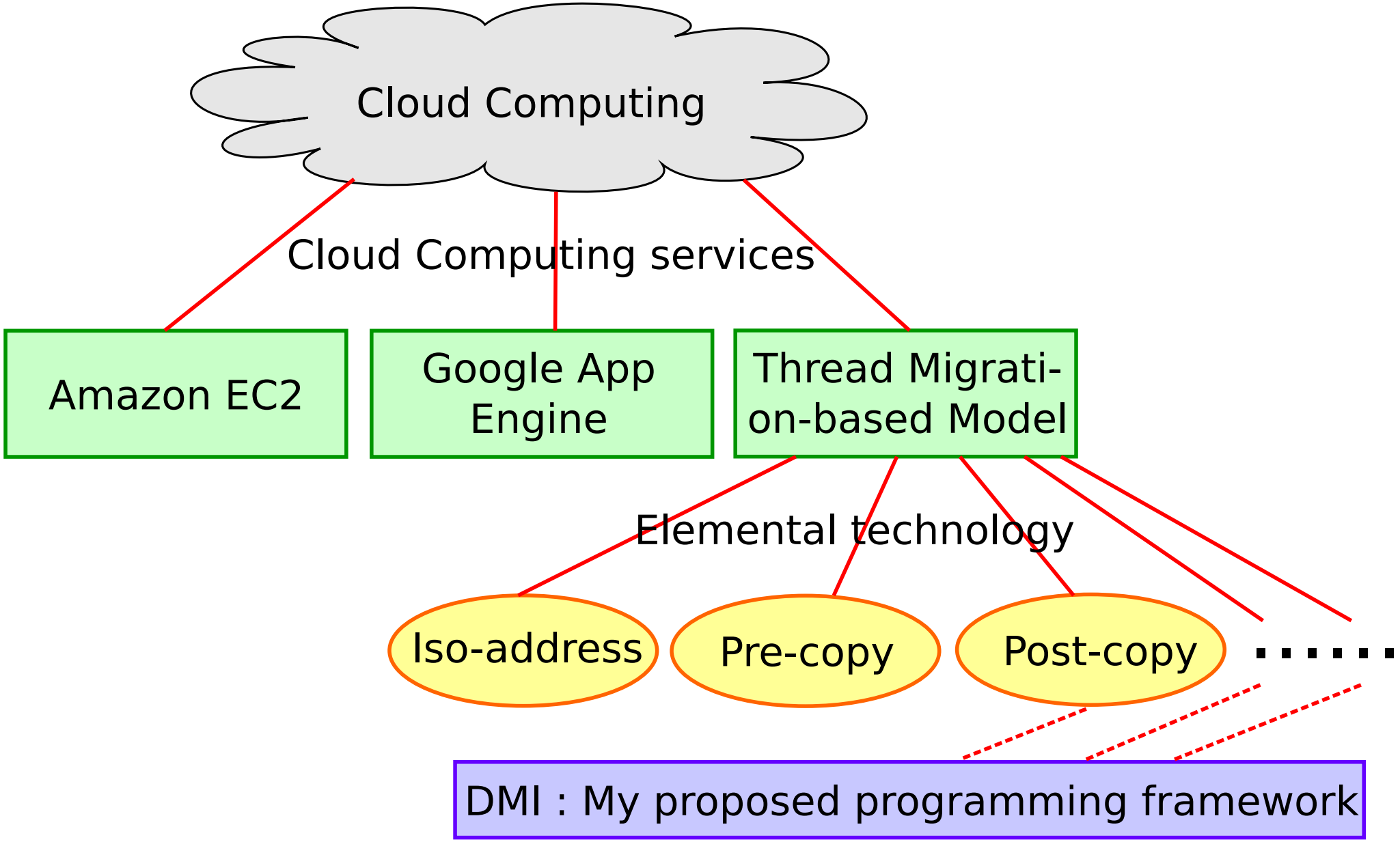
How does it schedule resources?(3)

- × Each app's performance depends on the global load condition
  - → Apps are distributed efficiently when the global load is low
  - → Apps are jammed when the global load is high



When the global load is low | When the global load is high

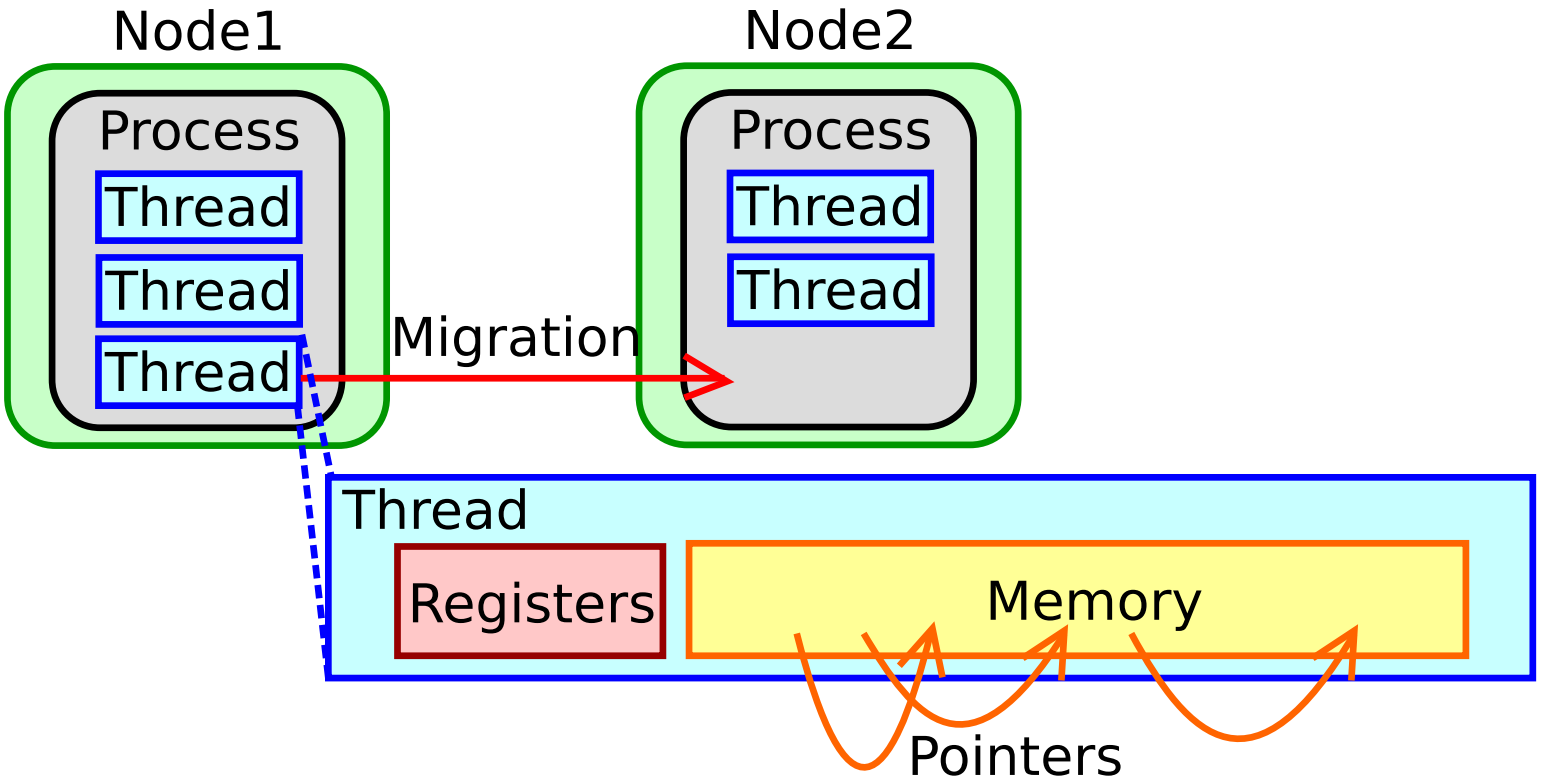
Road map


Cloud Computing
Cloud Computing services
Amazon EC2
Google App Engine
Thread Migrati-on-based Model
Elemental technology
Iso-address
Pre-copy
Post-copy
DMI : My proposed programming framework

3. Kernel Thread Migration

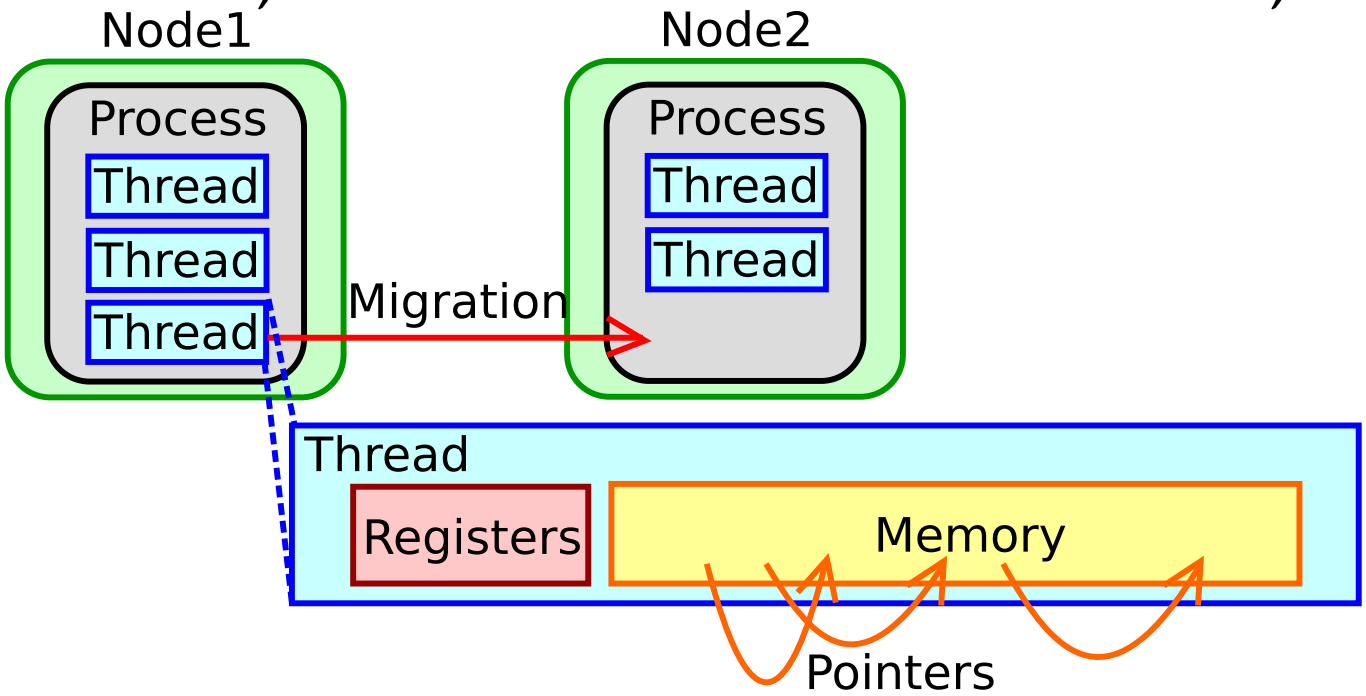
What is kernel thread migration?

- To migrate a running kernel thread from a process on a source node to a process on another node
- The entity of a kernel thread = CPU registers + memory(=stack + heap)
- Memory includes pointers



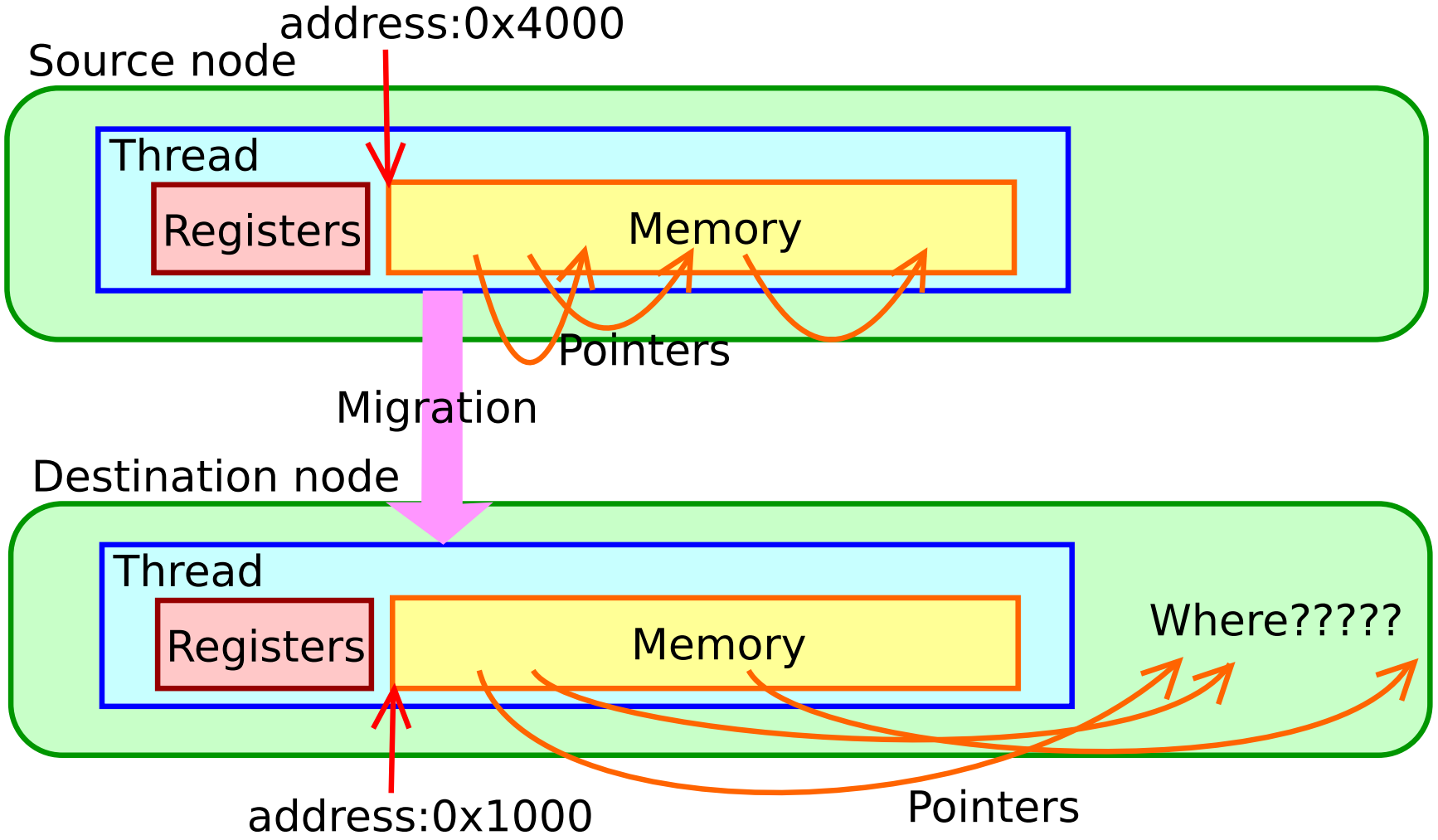
Assumptions

- Each process has multiple threads
- Each thread just accesses memory of the thread
  - → A thead does not access another thread's memory
    - ♦ Data sharing between threads is achieved by DSM layer
  - → No file access, no network communication, ...



A problem : Pointer invalidation

- A pointer is invalidated unless memory is located at the exact same address on a destination node as on a source node



Two solutions for pointer invalidation

(1) △ Updating all pointers correctly on a destination node

➔ It is hard to do perfectly, since an address value is sometimes indistinguishable from an integer value in C[Cronk et al,1997]

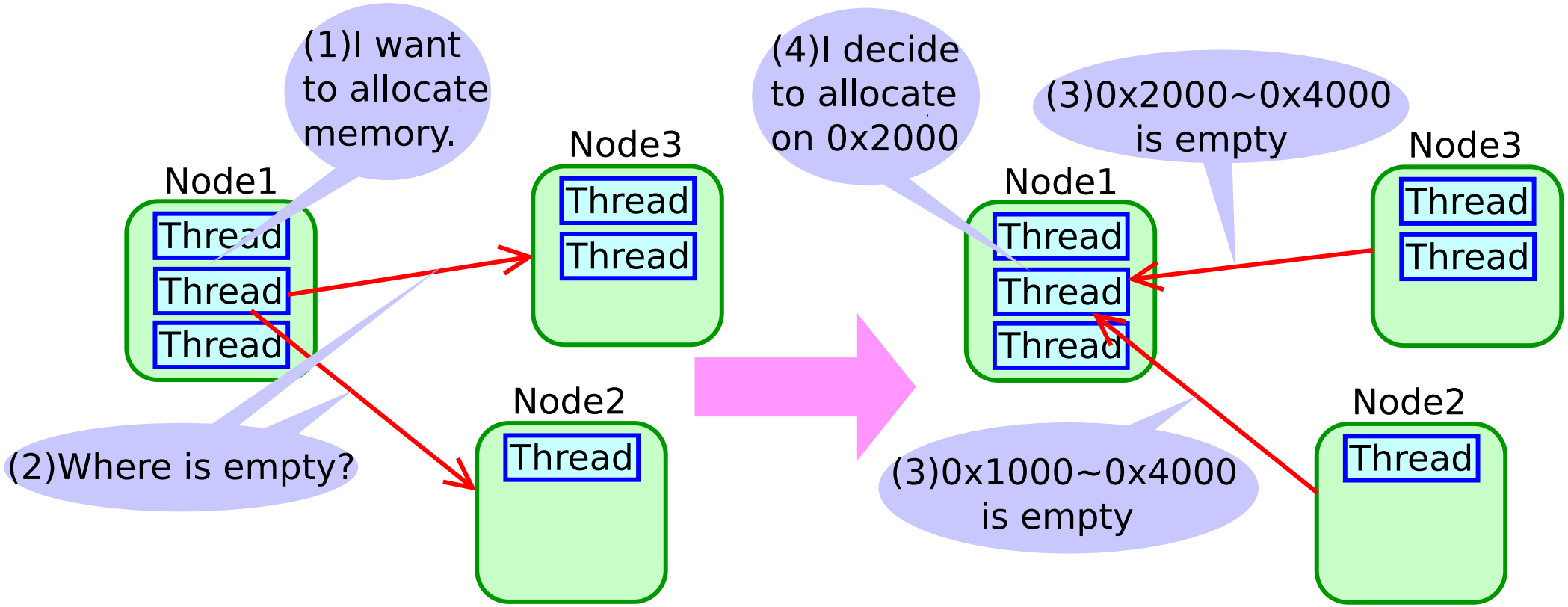
(2) ◎ Guaranteeing that the address space allocated on one thread is never allocated on any other threads

➔ This enables thread migration to the exact same address

➔ Iso-address[Antoniu, 1999]

How to achieve such allocation naively

- Negotiating where to allocate memory with all nodes at every memory allocation
  - → Too inefficient!

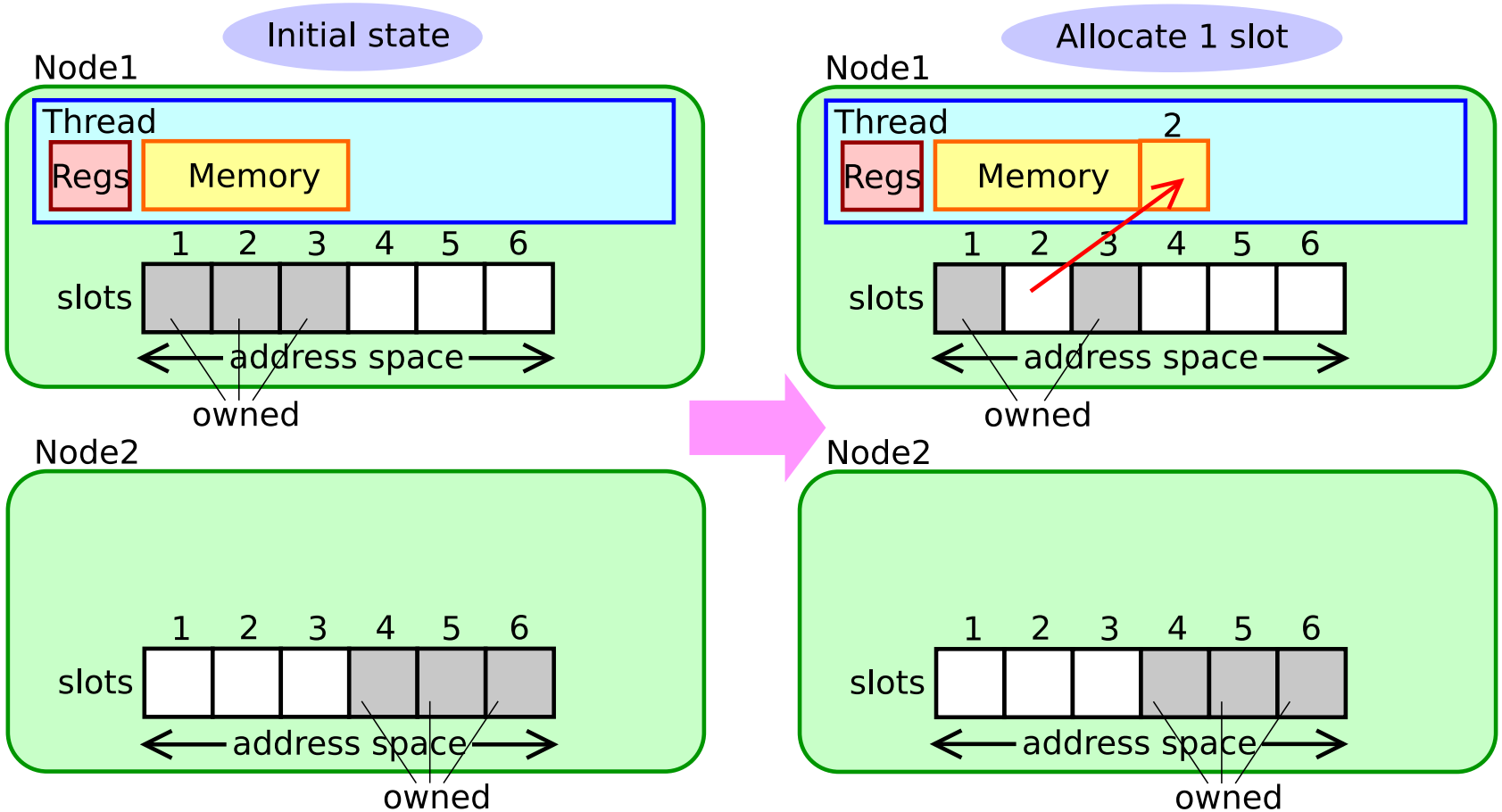


More refined approach : Iso-address(1)

(1) Address space is divided into "slots"

(2) Initially, distribute the whole slots to all nodes

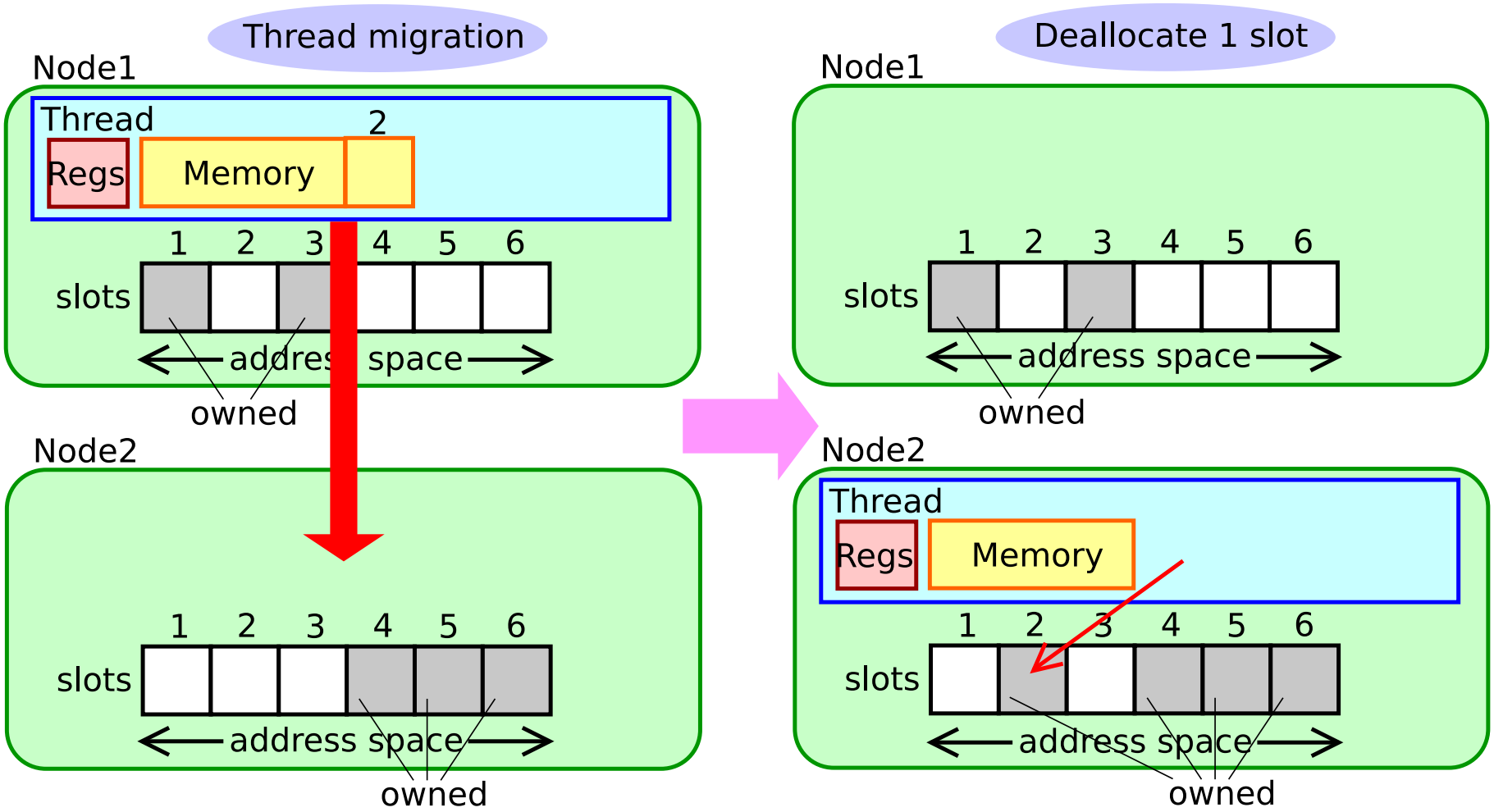
(3) Thread *t* allocates memory on the slots owned by *t*'s belonging node without inter-node communication



More refined approach : Iso-address(2)

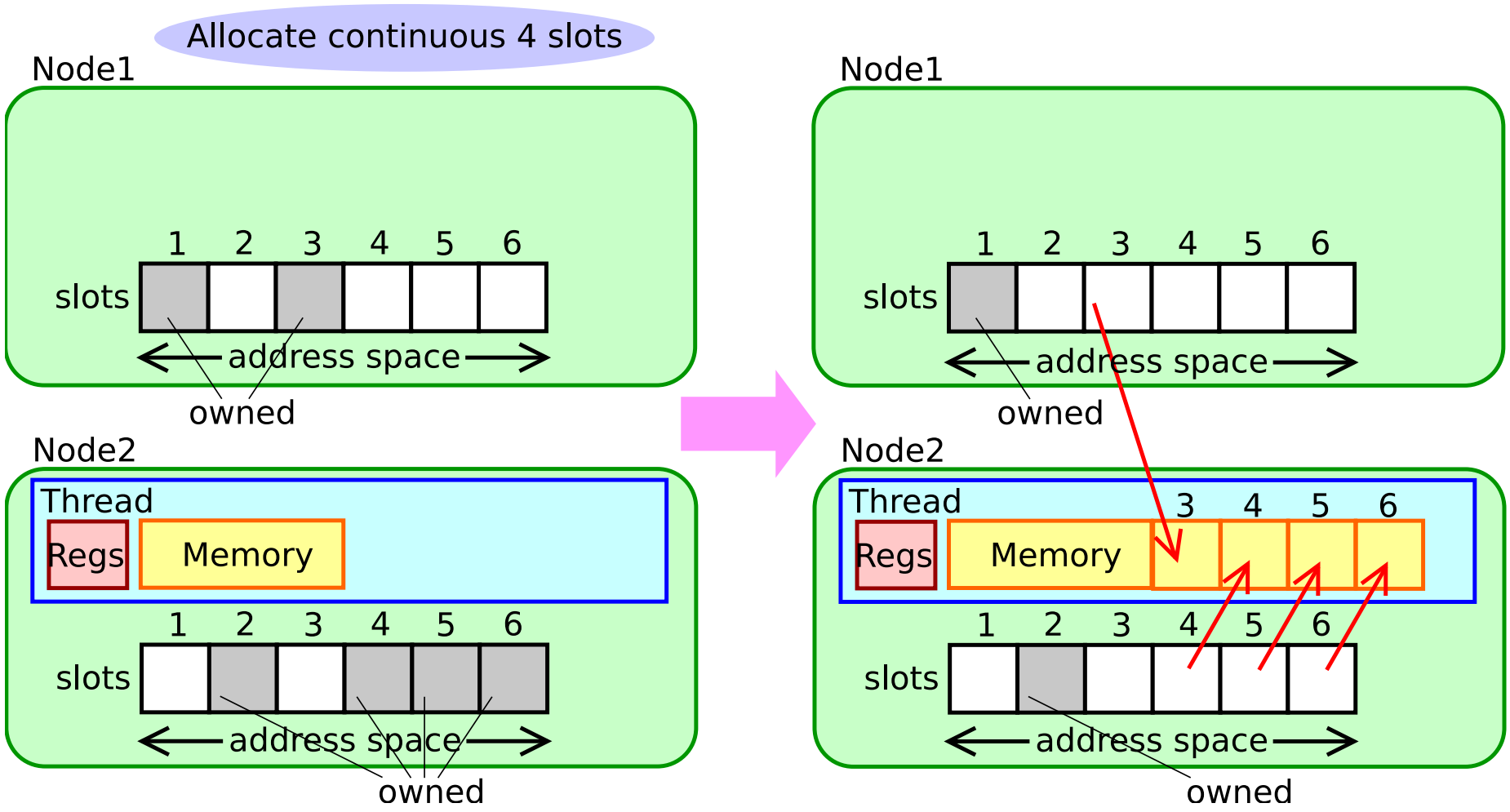
(4) Thread *t* can migrate to the same address of another node

(5) Thread *t* deallocates memory and releases the slots to *t*'s belonging node



More refined approach : Iso-address(3)

(6) When lacking in slots, a node steals some slots from another node



4. Fast Memory Migration

A problem of naive thread migration

- Iso-address enables thread migration as follows :
  (1) Stop the thread on a source node
  (2) Migrate CPU registers and memory
  (3) Resume the thread on a destination node
- A problem : Downtime is too long if the thread has huge memory

Fast Memory Migration

- Techniques for downtime reduction :

  (1) Pre-copy[Clark et al, 2005]

  - Migrate memory before thread migration

  (2) Post-copy[Hines et al, 2009]

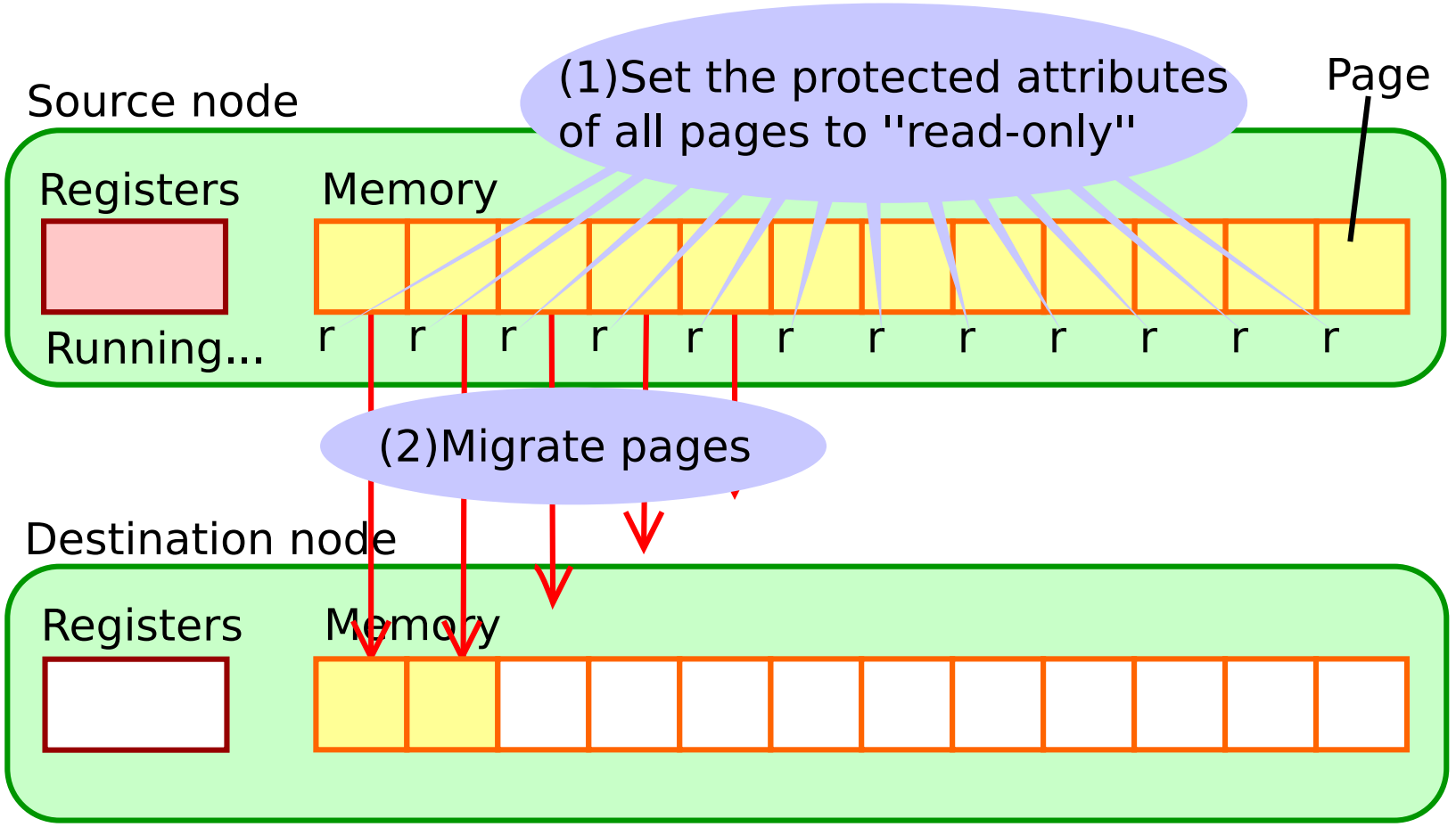
  - Migrate memory after thread migration

Pre-copy : Round 1(1)

(1) Forbid write access to all pages

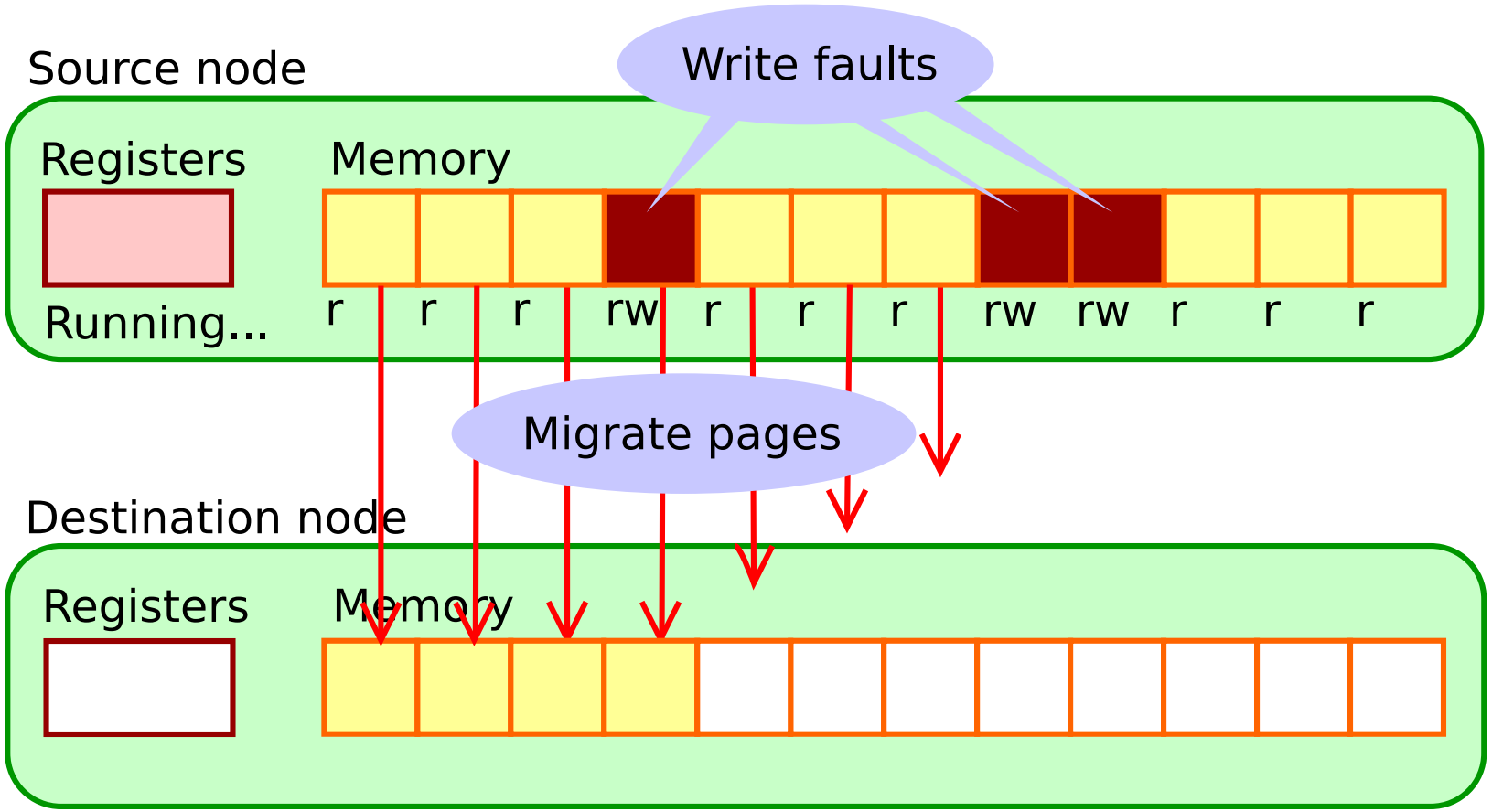
(2) Migrate all pages in a background process

➔ During this time, the thread runs concurrently on a source node



Pre-copy : Round 1(2)

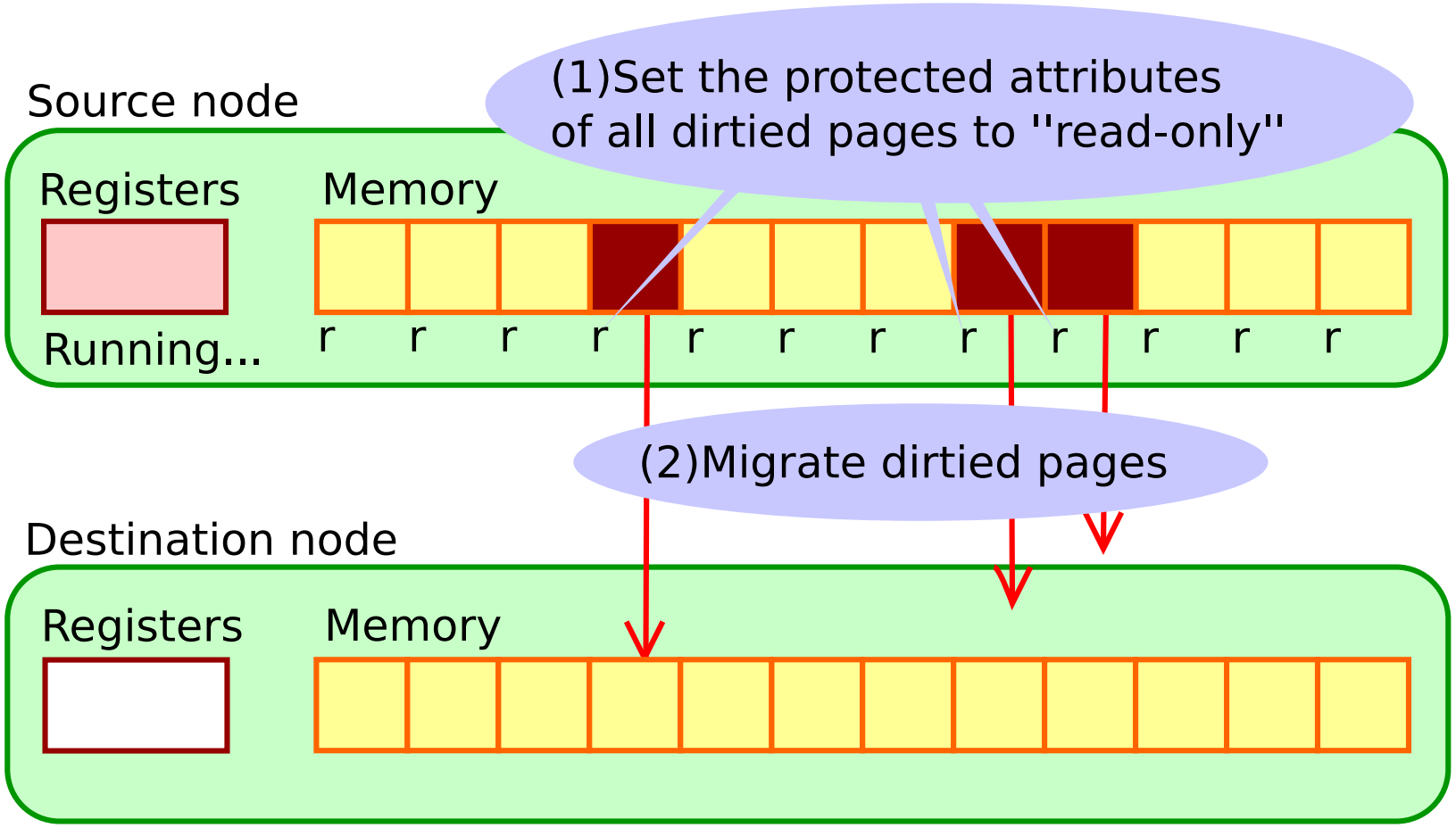
(3) Detect and record write faults of the thread



Pre-copy : Round 2

(4) Again, forbid write access to all pages

(5) Migrate the pages dirtied during the round 1



Pre-copy : Round n

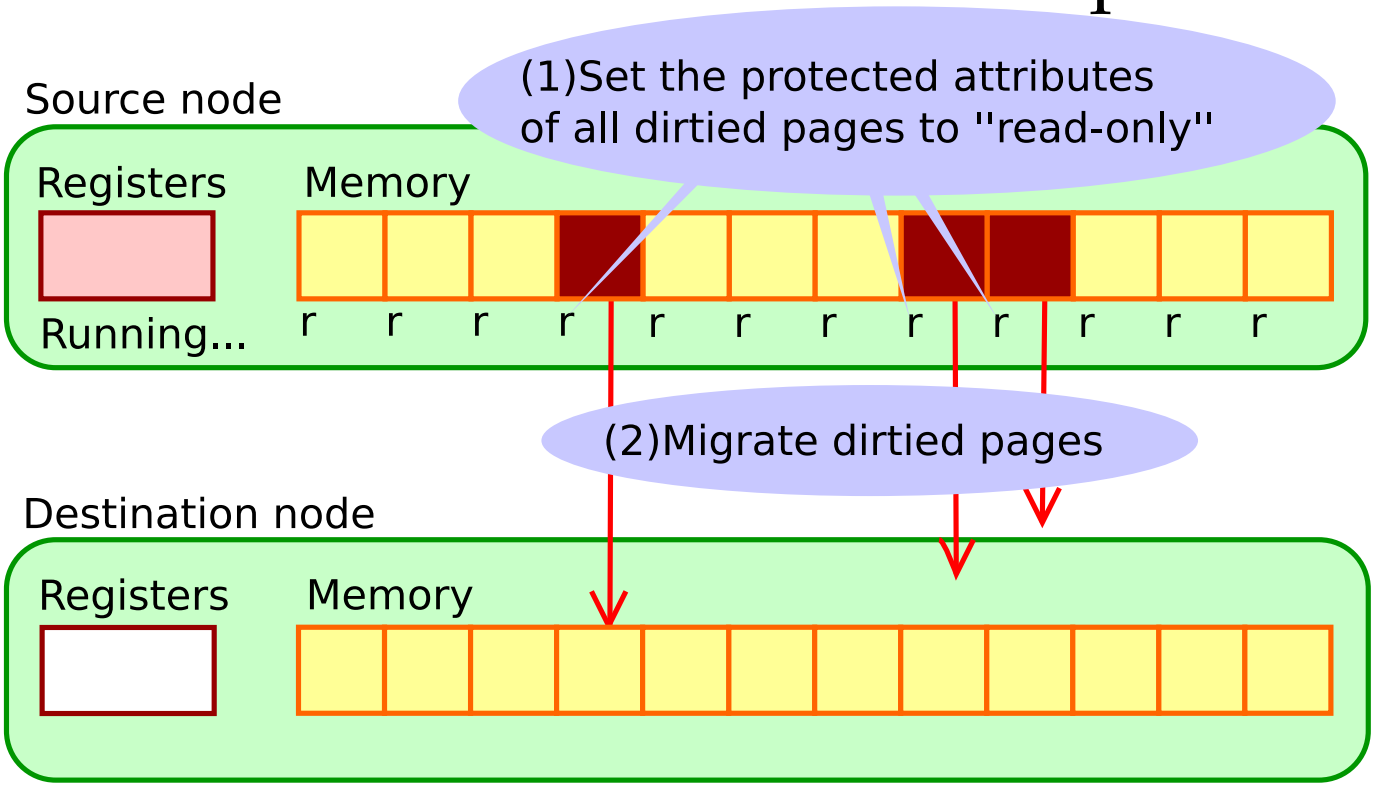
(6) Again, forbid write access to all pages

(7) Migrate the pages dirtied during the round $n-1$

(8) Repeat such rounds until

➔ the number of dirtied pages becomes small

➔ or the number of rounds exceeds the predefined limit

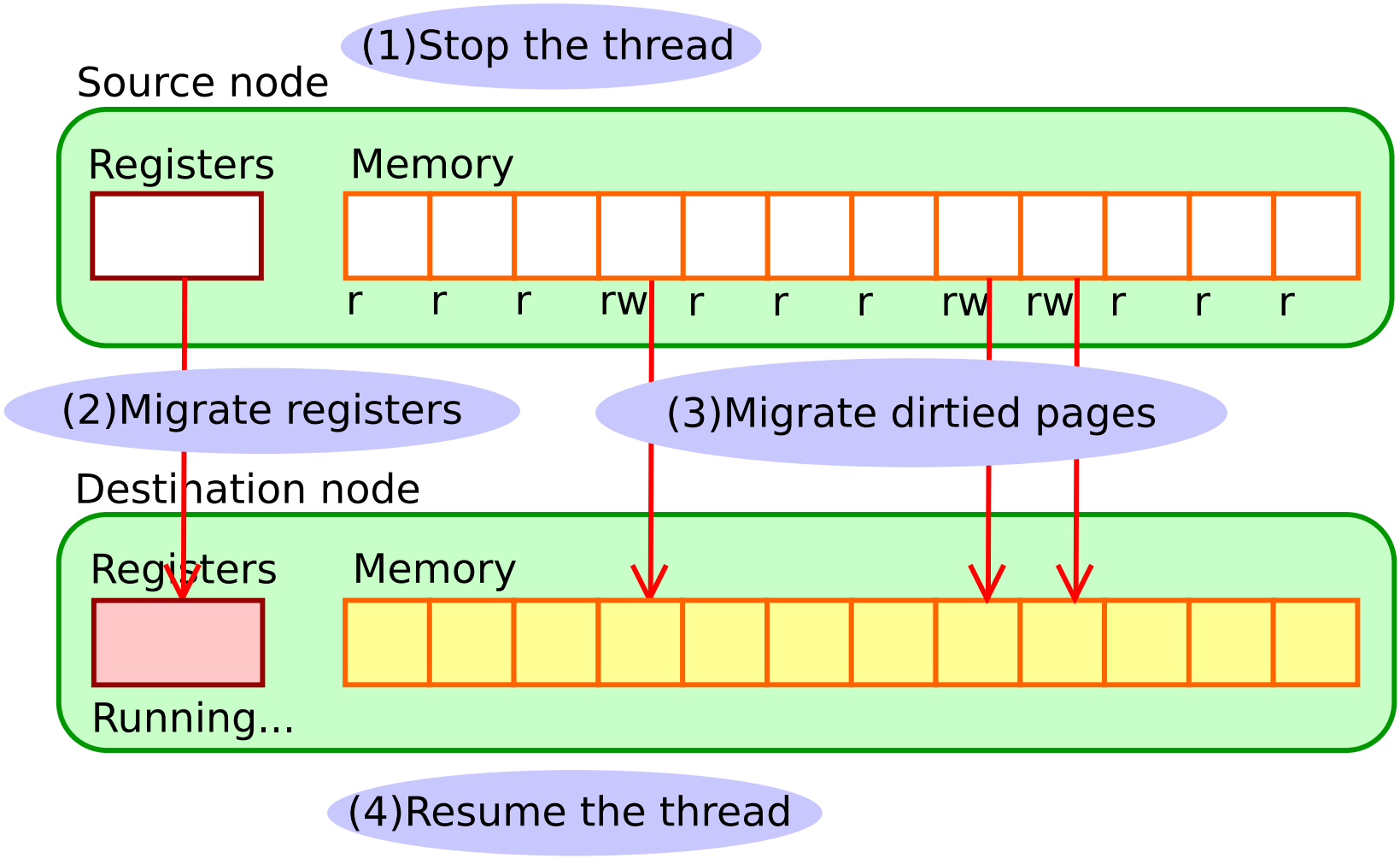


Pre-copy : The final copy

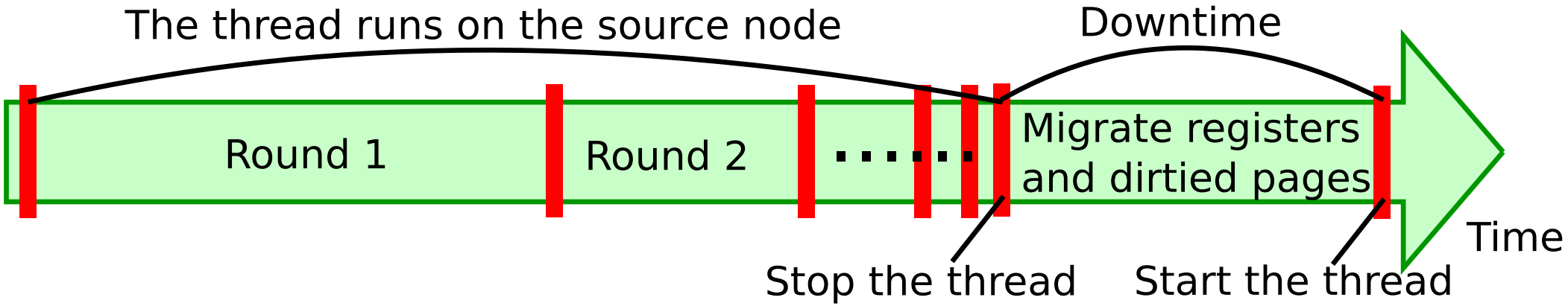
(9) Stop the thread on a source node

(10) Migrate CPU registers and all dirtied pages

(11) Resume the thread on a destination node



Pre-copy : A time-line



Pre-copy : An improvement

- A motivation : Migrating dirtied pages many times should be avoided
- An observation : Memory access has temporal locality
  - → The pages frequently dirtied during the previous rounds will be again dirtied in the near future
- An improvement :
  - → In the round n, migrate only the pages dirtied during the round $n-1$ that have not frequently been dirtied during the previous rounds

Pre-copy : Characteristics

- × Dirtied pages are migrated many times
- × The number of migrated pages is app-dependent
  - → Read-intensive apps : The number of migrated pages ≳ The number of actually used pages
  - → Write-intensive apps : The number of migrated pages > The number of actually used pages
- × Downtime is long especially for the write-intensive apps
- ◎ Running apps' performance degrades little since pages are migrated in a background process

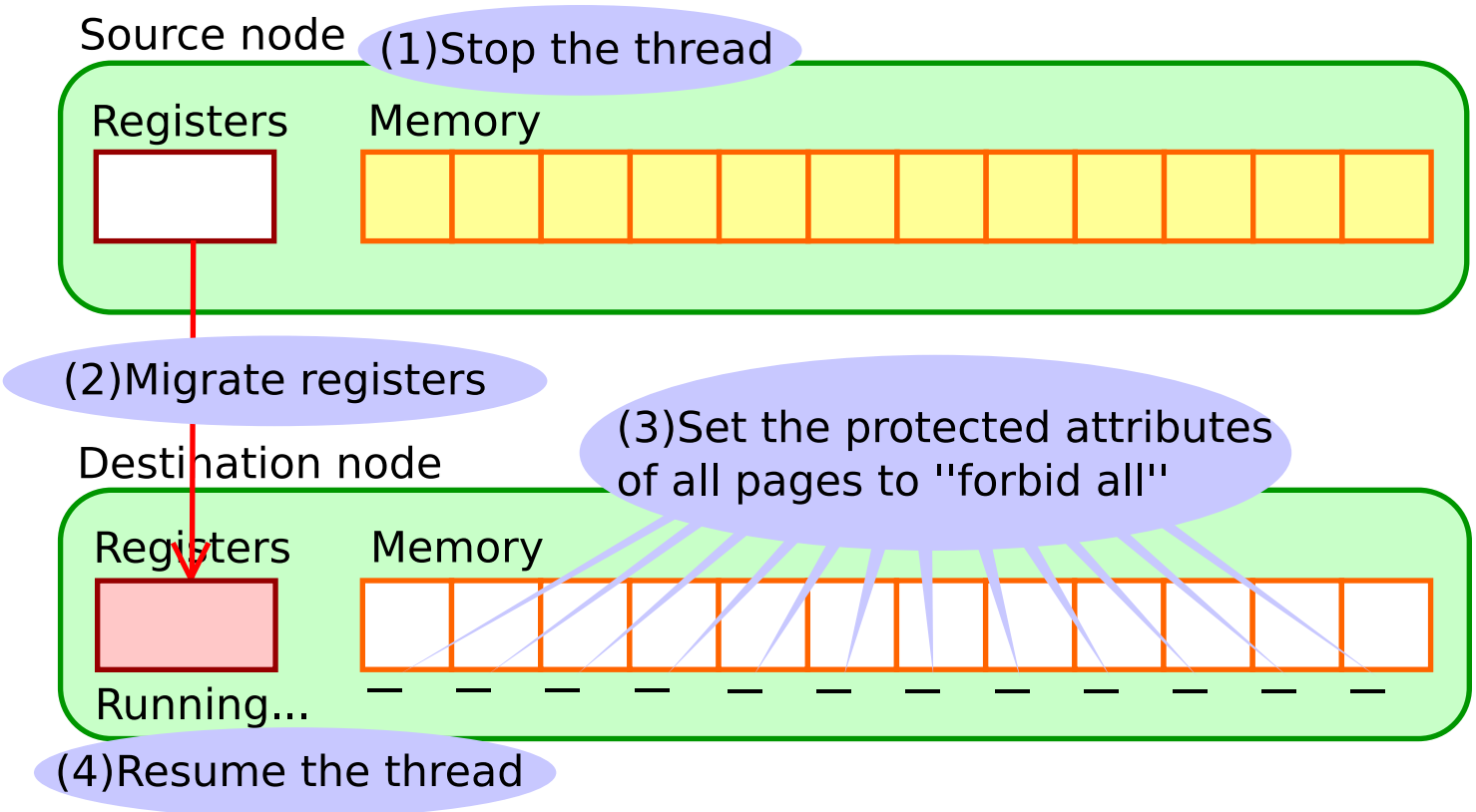
Post-copy : The basic idea(1)

(1) Stop the thread on a source node

(2) Migrate only CPU registers

(3) Forbid any read/write access to all pages on a destination node

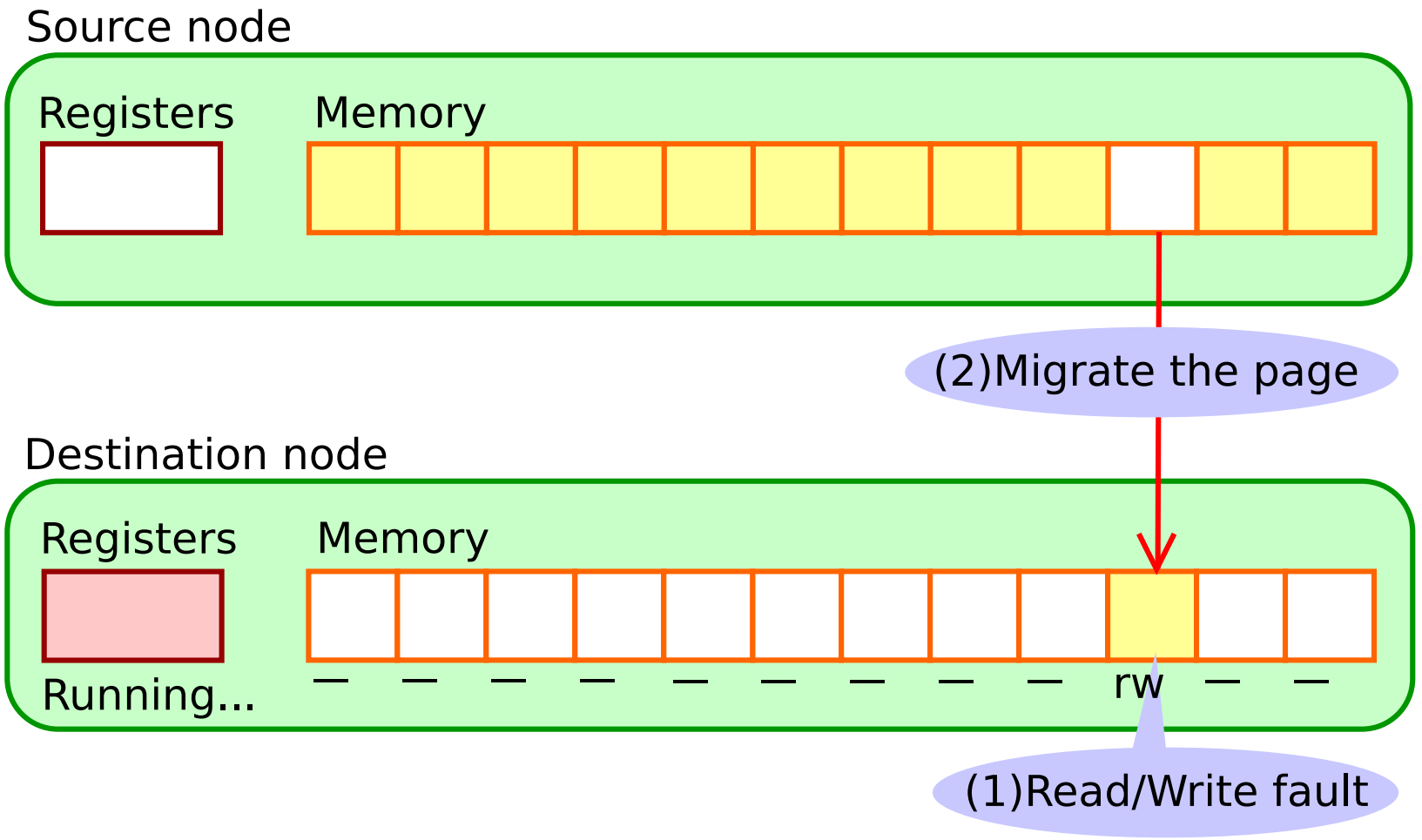
(4) Resume the thread on a destination node



Post-copy : The basic idea(2)

(5) Detect a read/write fault of the thread

(6) Migrate the page in a demand-driven manner



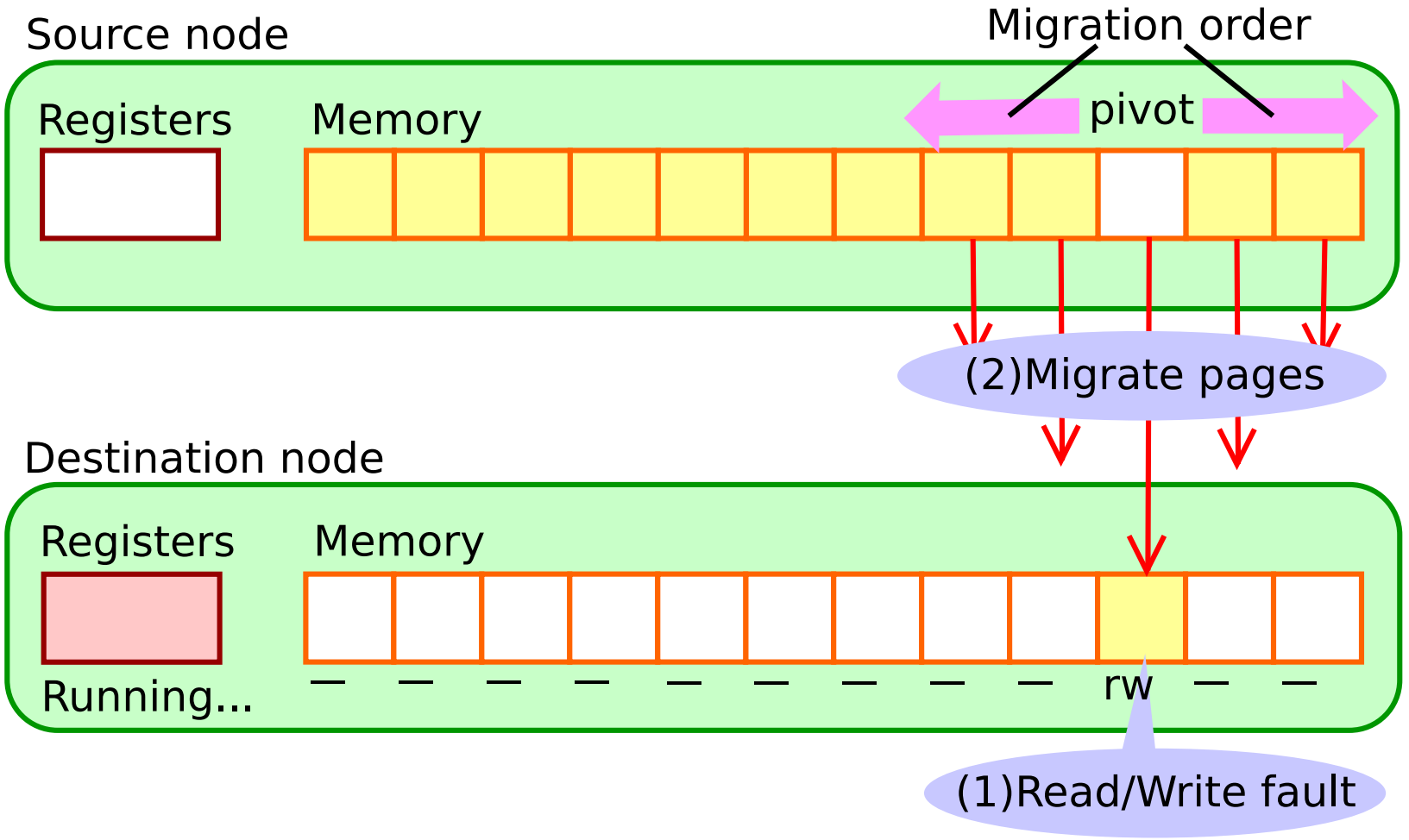
Post-copy : Problems of the basic idea

- Problems :
  - → Running apps' performance degrades at every read/write fault
  - → Some pages remain on a source node unless the thread accesses them
- A solution : Forcing a background process to migrate pages, considering temporal access locality

Post-copy : Improvements(1)

(7) *pivot*=the page on which the thread caused a read/write fault most recently

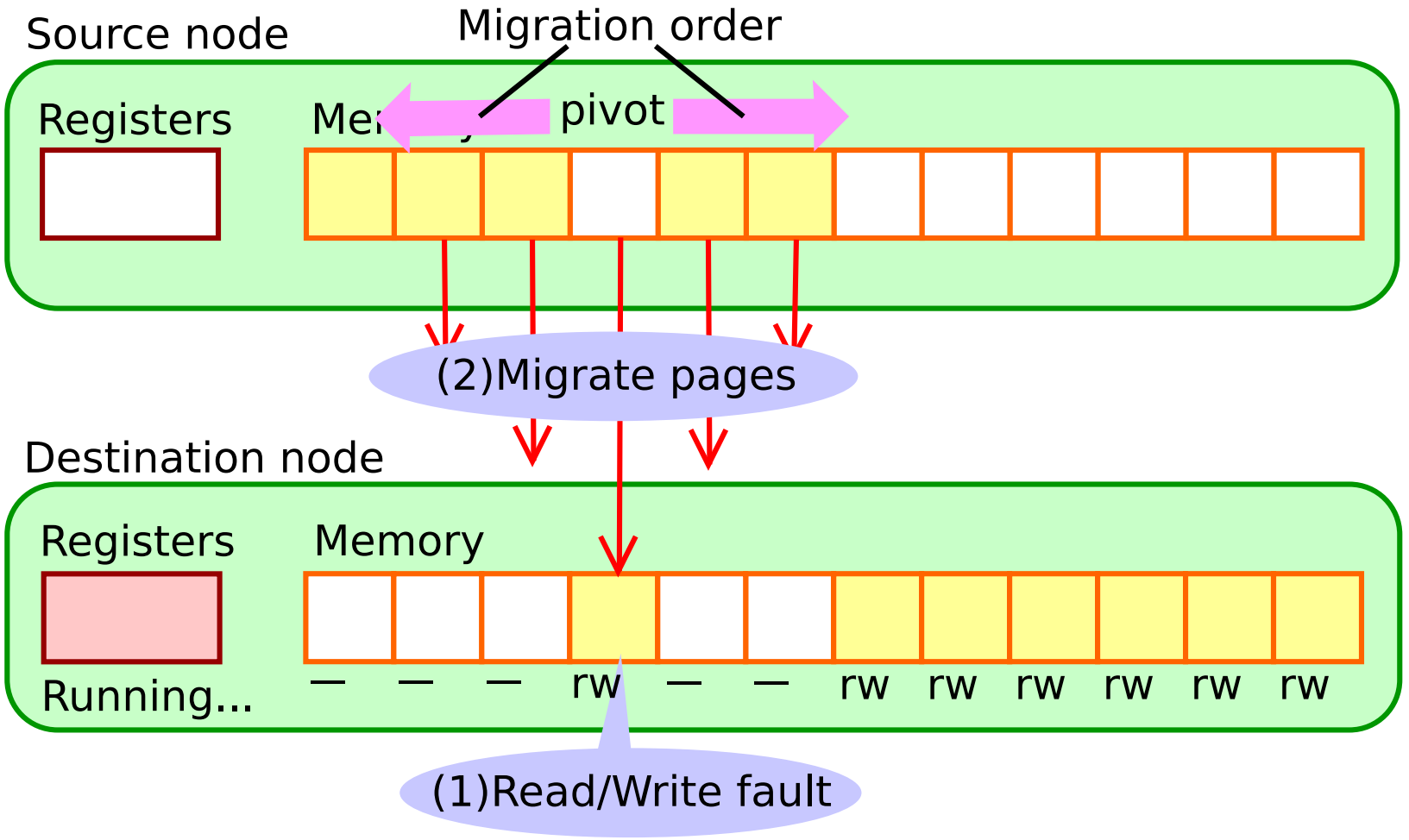
(8) A background process migrates pages around the *pivot*



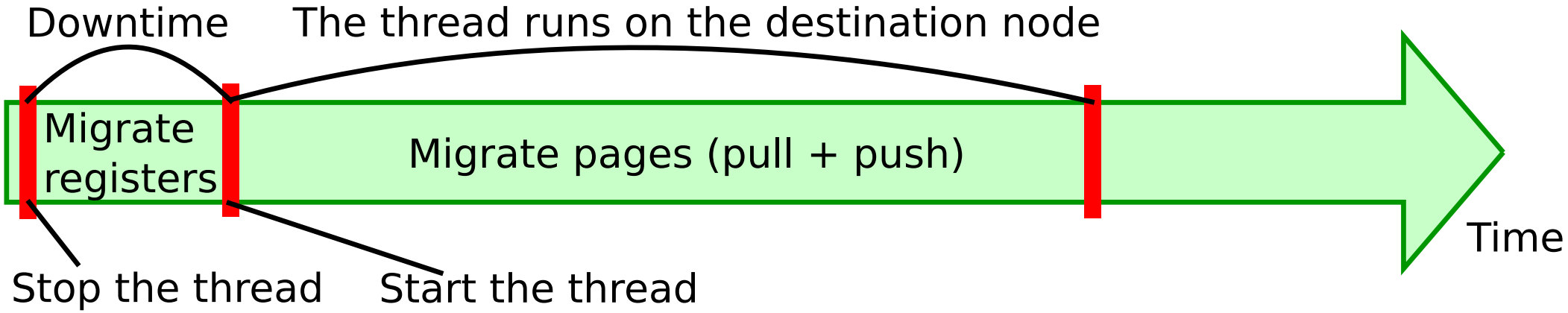
Post-copy : Improvements(2)

(9) The *pivot* is updated at every read/write fault

(10) A background process continues to migrate pages around the *pivot*



Post-copy : A time-line



Post-copy : Characteristics

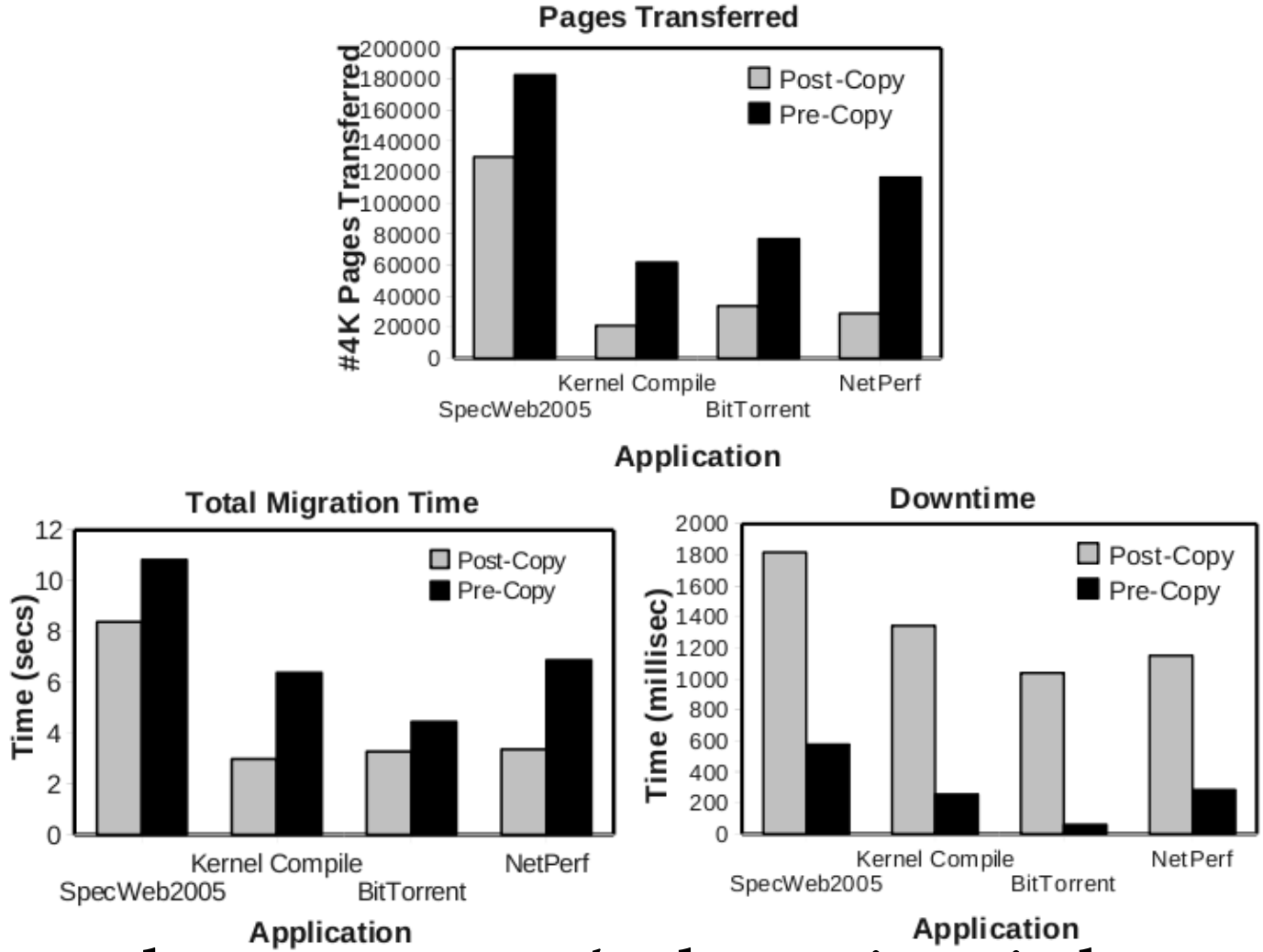
- ◎ Stable for broad range of apps
  - → For both read-intensive and write-intensive apps : The number of migrated pages = The number of actually used pages
- ◎ Downtime is short
  - → because all that have to be migrated during downtime are CPU registers
- × Running apps' performance degrades at every read/write fault

Pre-copy vs Post-copy : A general view

- The number of migrated pages
  - → Read-intensive apps : Pre-copy $\gtrsim$ Post-copy
  - → Write-intensive apps : Pre-copy $\gg$ Post-copy
- Stability for broad range of apps
  - → Pre-copy $<$ Post-copy
- Downtime
  - → Pre-copy $>$ Post-copy
- Running apps' degradation
  - → Pre-copy $<$ Post-copy

Pre-copy vs Post-copy : Experimental results

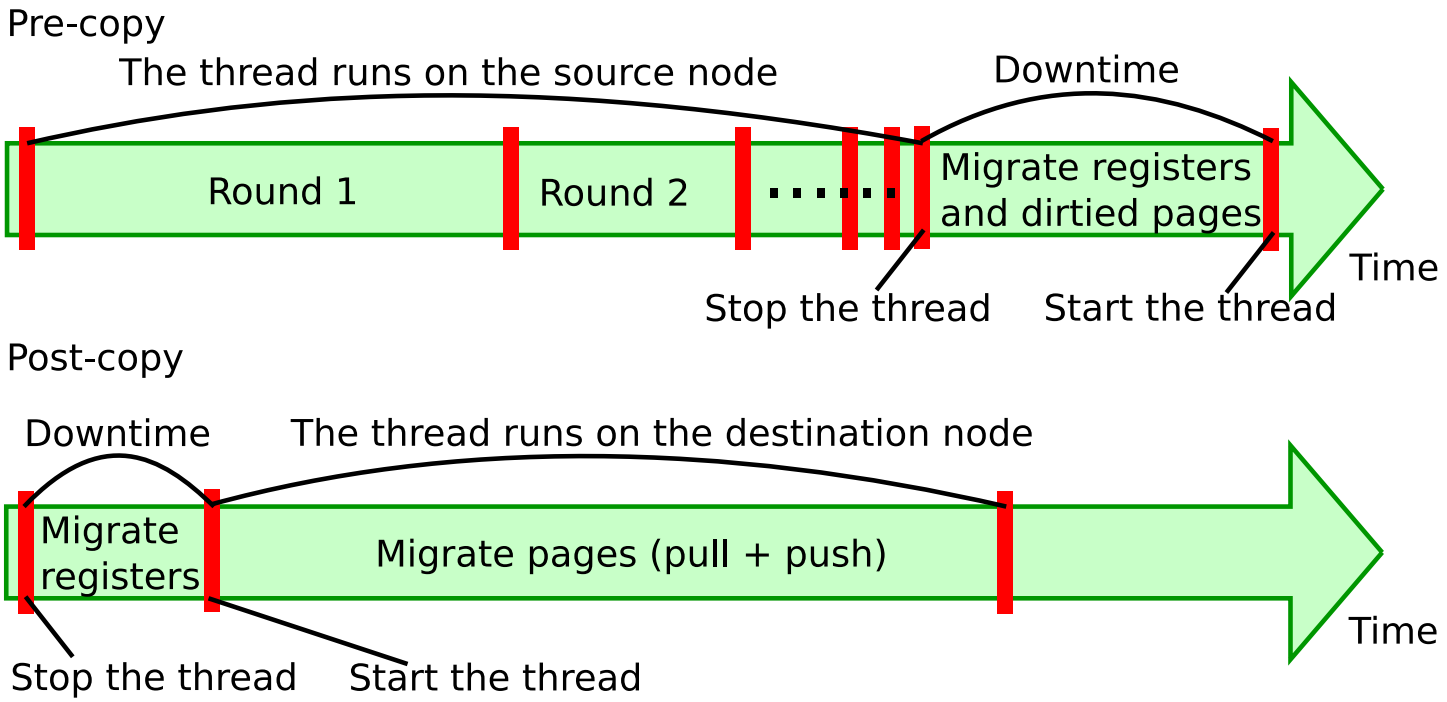
- Pre-copy vs Post-copy in VM migration[Hines et al, 2009]



- The reason that post-copy's downtime is longer than pre-copy's downtime is due to implemental issues

Pre-copy vs Post-copy : In our scenario

- Rapid Adaptability to load fluctuation is important in the thread migration-based model
  - → A running thread should be migrated immediately when needed
- Post-copy is more suitable than Pre-copy



5. Conclusions

Conclusions(1)

- Common requirements for Cloud Computing services :
  - (1) To support flexible scale-up/scale-down in response to load increase/decrease
  - (2) To schedule shared resources between users (according to some policies)
- Three Cloud Computing services :

	Amazon EC2	Thread migration-based Model	GAE
Unit of scale-up/scale-down	VM	Thread	Request
Resource consumption	Large	Middle	Small
Billing granularity	Coarse	Middle	Fine
Adaptability to load fluctuation	Slow	Middle	Rapid
Domain of targeted apps	Large	Middle	Small
Long-time apps	OK	OK	NG

Conclusions(2)

- How to achieve the thread migration-based model
  - → Kernel thread migration
    - ♦ Iso-address enables memory allocation with little inter-node communication
  - → Fast memory migration
    - ♦ Post-copy is more suitable than Pre-copy for the thread migration-based model