# Yabusame:
# Postcopy Live migration for QEmu/KVM

* Isaku Yamahata, VALinux Systems Japan K.K. <yamahata@private.email.ne.jp>
  Takahiro Hirofuchi, AIST <t.hirofuchi@aist.go.jp>

LinuxConJapan June 7th, 2012

# Agenda

- Demo
- Precopy vs Postcopy
- Implementation
- Evaluation
- Future work
- Summary

From wikipedia

Yabusame is a joint project with Takahiro Hirofuchi, AIST and Satoshi Itoh, AIST.
This work is partly supported by JST/CREST ULP and KAKENHI (23700048).
The development of Yabusame was partly funded by METI (Minister of Economy, Trade and Industry) and supported by NTT Communications Corporation.

# Demo

A VM with 1GB RAM is live-migrated to the right PC.

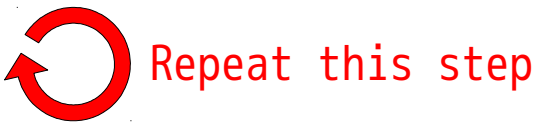
Normal Live Migration

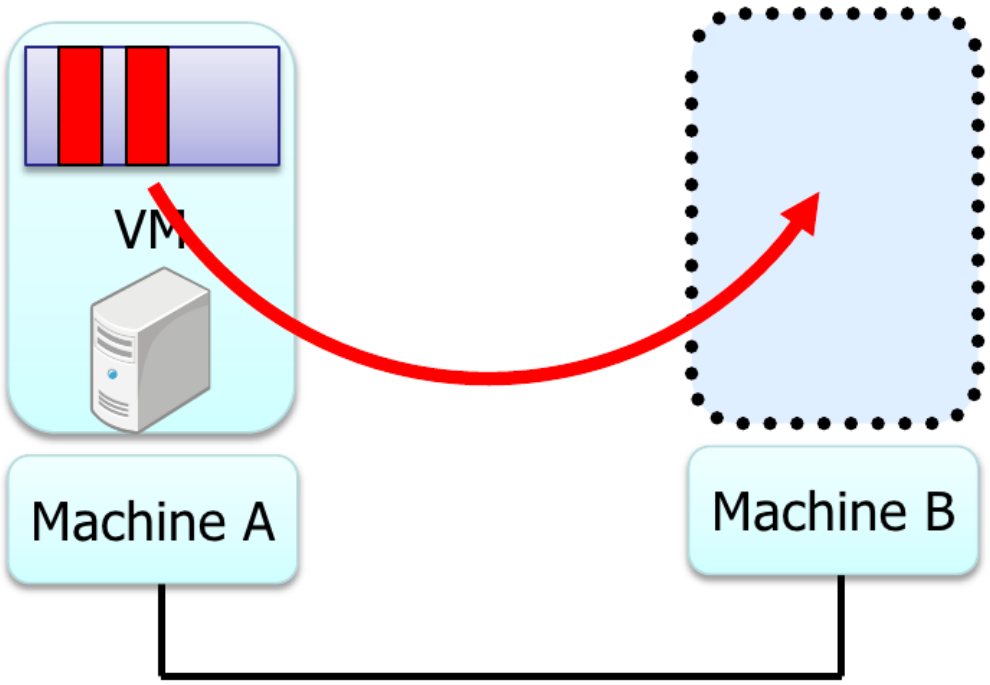
Yabusame Live Migration (Developed by AIST)

# Precopy vs Postcopy

# Precopy live migration

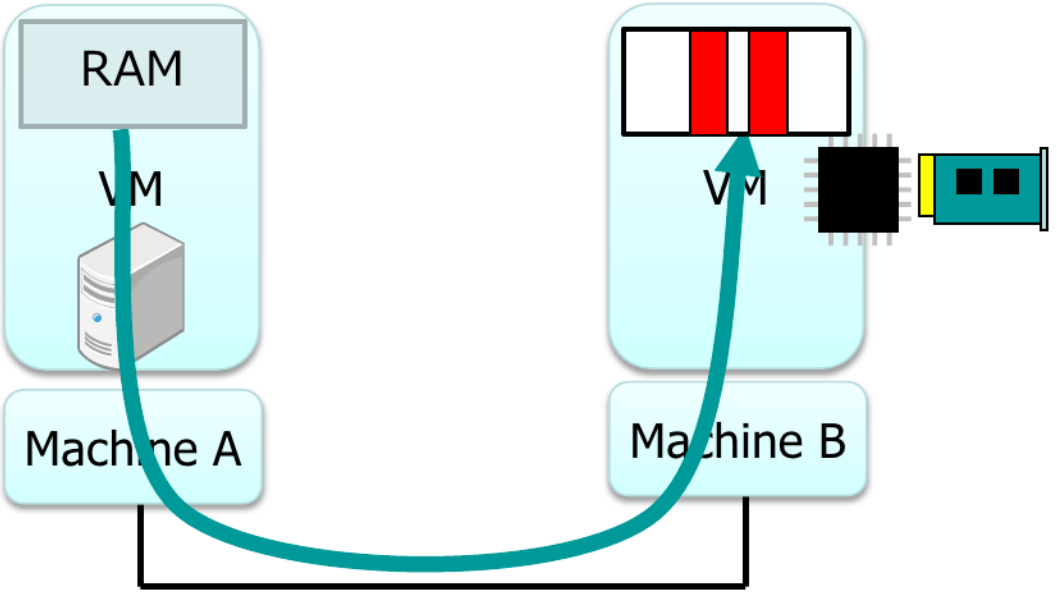
1. Enable dirty page tracking
2. Copy all memory pages to destination
3. Copy memory pages dirtied during the previsou copy again
4. Repeat the 3rd step until the rest of memory pages are enough small.
5. Stop VM
6. Copy the rest of memory pages and non-memory VM states
7. Resume VM at destination



# Postcopy live migration

1. Stop VM
2. Copy non-memory VM states to destination
3. Resume VM at destination
4. Copy memory pages on-demand/backgroundly
   - Async PF can be utilized

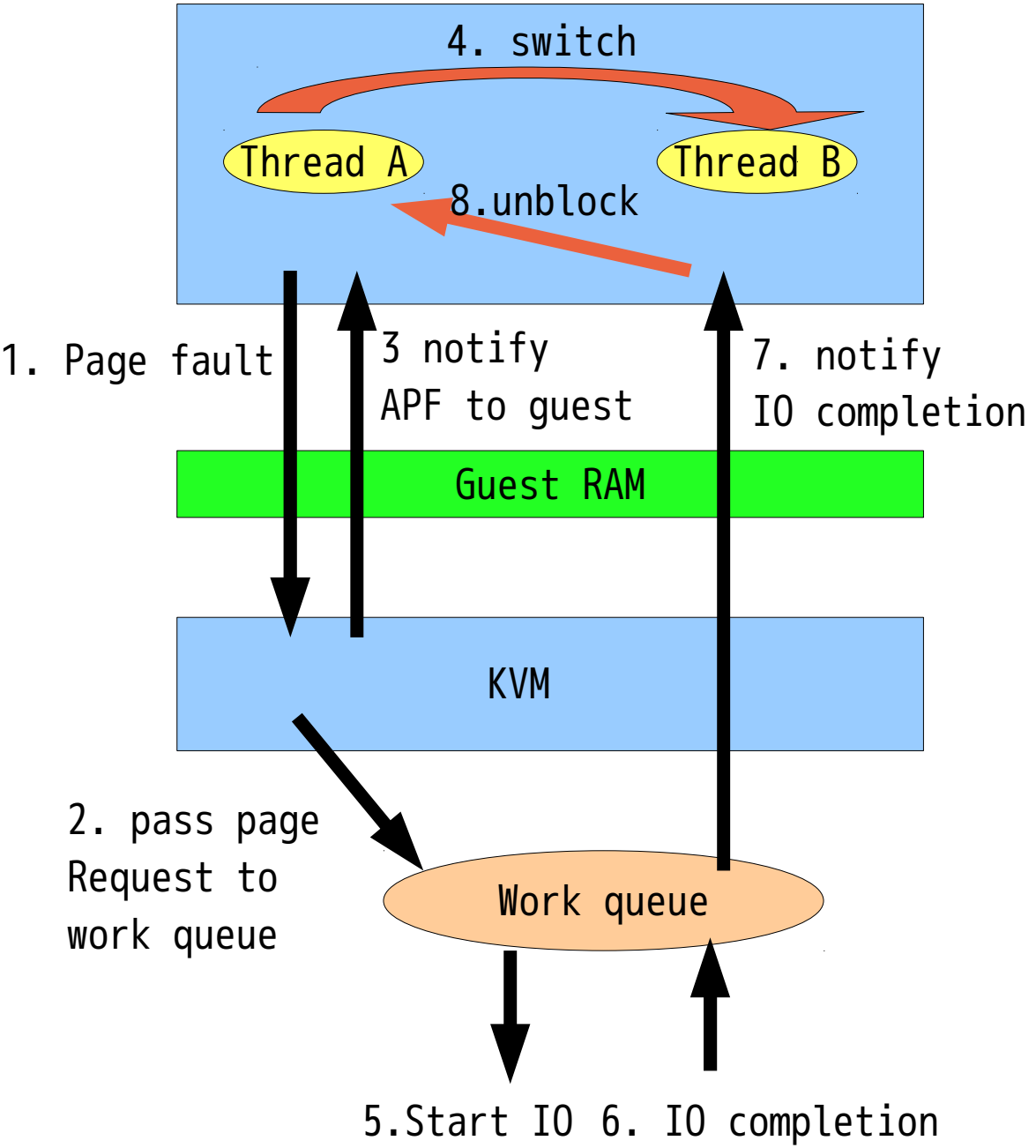



Copy memory pages
- On-demand(network fault)
- background(precache)

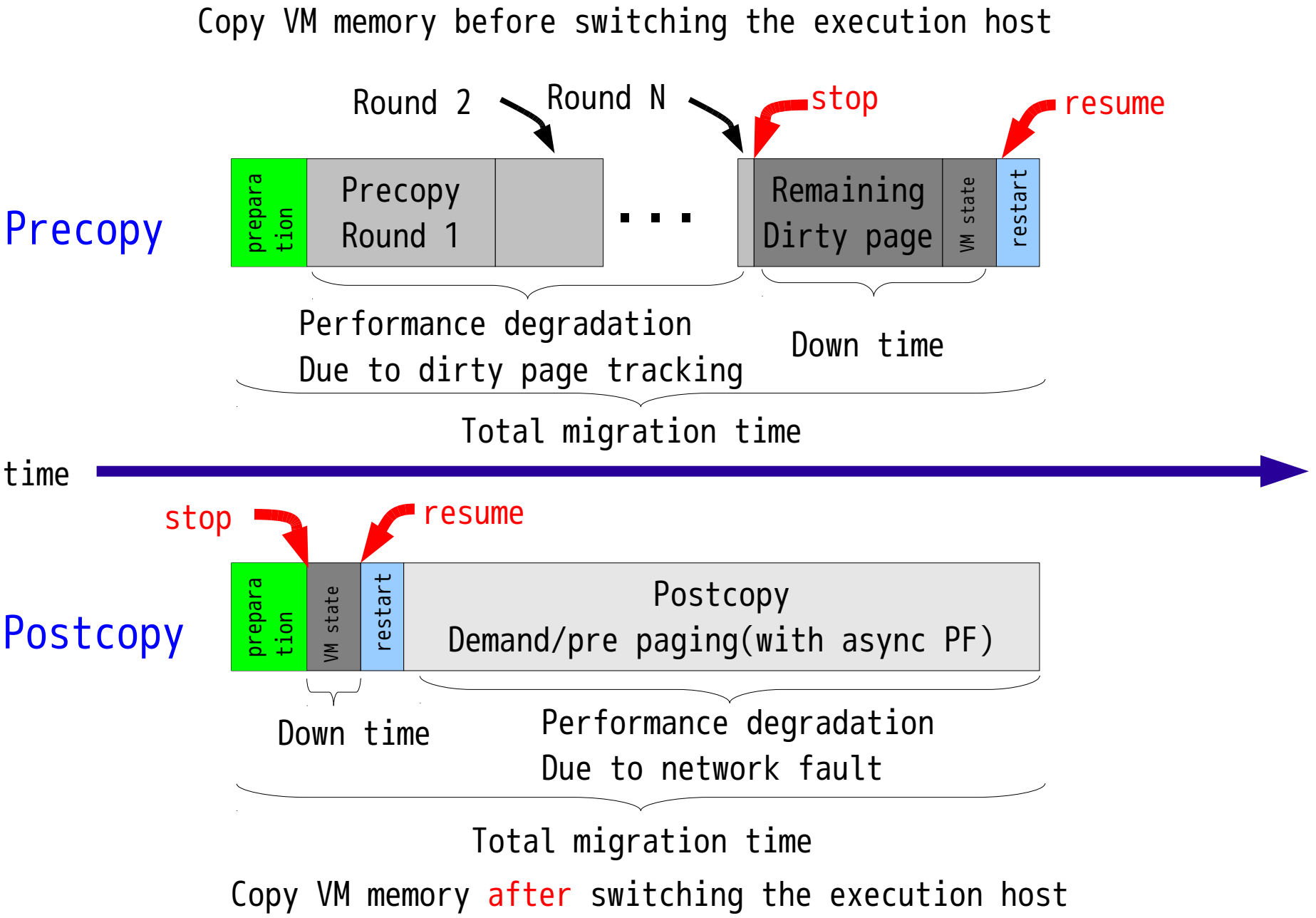
# Asynchronous Page Fault(APF)

AFP can be utilized for postcopy

1. Guest RAM as host page can be swapped out(or on the src machine in case of postcopy)
2. When the page is faulted, worker threads starts IO
3. Notify it to (PV) guest
4. Guest blocks the faulting thread and switches to another thread
5. When IO is completed, it is notified to guest
6. Unblock the previously blocked thread



# Total migration/down time

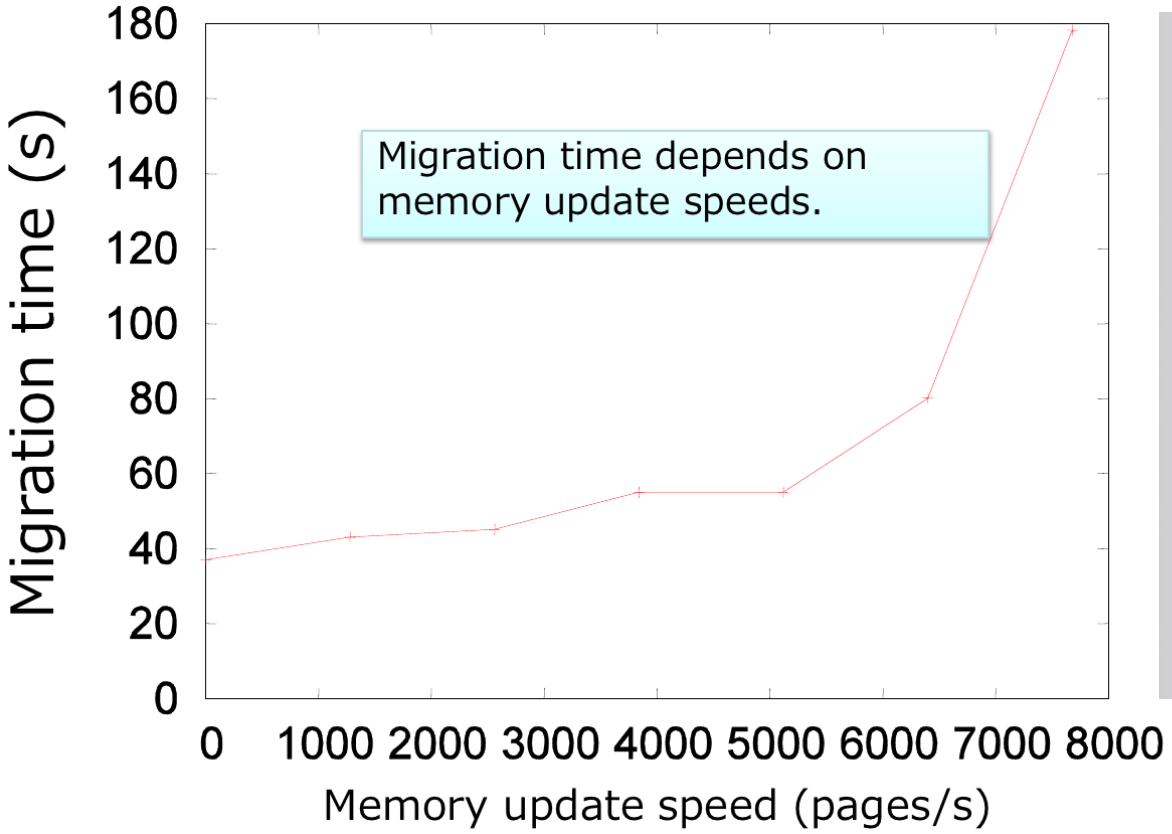


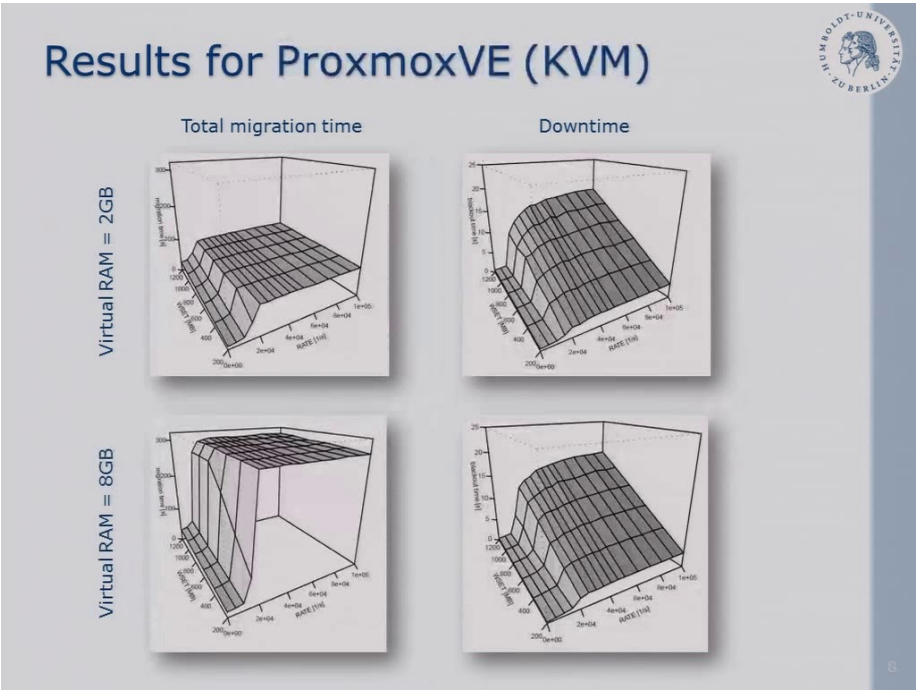
# Precopy vs Postcopy

| | precopy | postcopy |
|---|---|---|
| Total migration time | (RAM size / link speed) +<br>overhead +<br>Non-deterministic<br>Depending on precopy rounds | (RAM size / link speed) +<br>ovedrhead |
| Worst downtime | (VMState time) +<br>(RAM size/link speed)<br><br>The number of dirtied pages can be optimized by ordering pages sent in precopy phase | VMState time<br>Followed by postcopy phase<br><br>Postcopy time is limited by (RAM size/link speed)<br>Alleviated by<br>Prepaging + AsyncPF |

# Precopy live migration

- Total migration time and downtime depend on memory dirtying speed
  - Especially the number of dirty pages doesn't converge when dirtying speed > link speed




Downtime Analysis for Pro-Active Virtual Machine Live Migration : Felix Salfner
http://www.tele-task.de/archive/video/flash/13549/

# Postcopy is applicable for

- Planned maintenance
  - Predictable total migration time is important
- Dynamic consolidation
  - In cloud use case, usually resources are over-committed
  - If machine load becomes high, evacuate the VM to other machine promptly
    - Precopy optimization (= CPU cycle) may make things worse

# Postcopy characteristic

- network bandwidth
  - Postcopy transfer a page only once
  - LAN case: Not all network bandwidth can be used for migration
    - network bandwidth might be reserved
  - non-LAN case: Inter-Datacenter live-migration
    - L2 connectivity among datacenters with L2 over L3 has becoming common
    - VM migration over DCs as Disaster Recovery
- reliability
  - VM can be lost if network failure during migration occurs

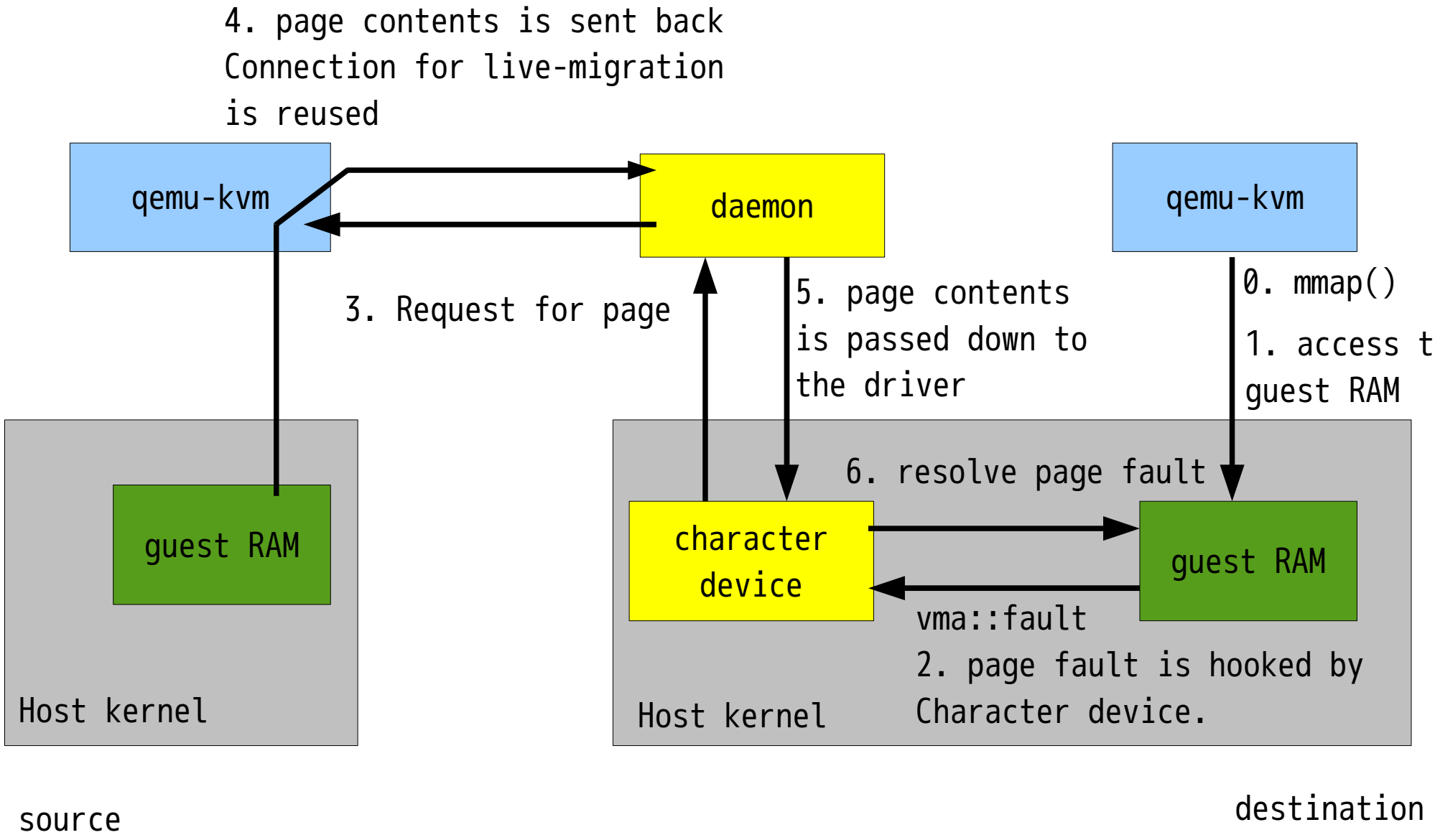
# Implementation

# Hooking guest RAM access

- Design choice
  - Insert hooks all accesses to guest RAM
  - Character device driver (umem char device)
    - Async page fault support
  - Backing store(block device or file)
  - Swap device

| | Pros | Cons |
|---|---|---|
| Modify VMM | portability | impractical |
| Backing store | No new device driver | Difficult future improvement<br>Some kvm host features wouldn't work |
| Character Device | Straight forward<br>Future improvement | Need to fix kvm host features |
| Swap device | Everything is normal after migration | Administration<br>Difficult future improvement |

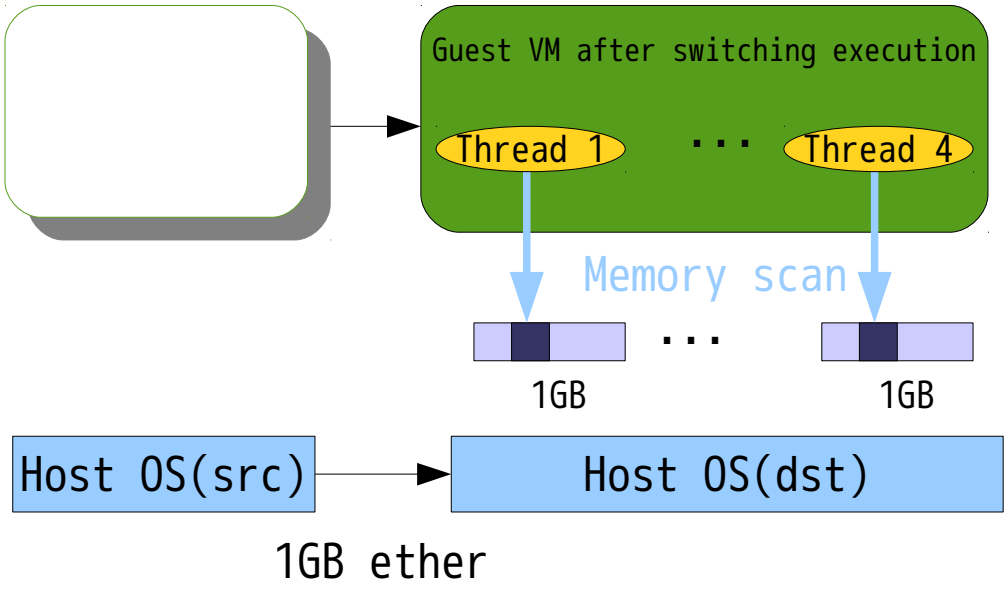
# Implementation



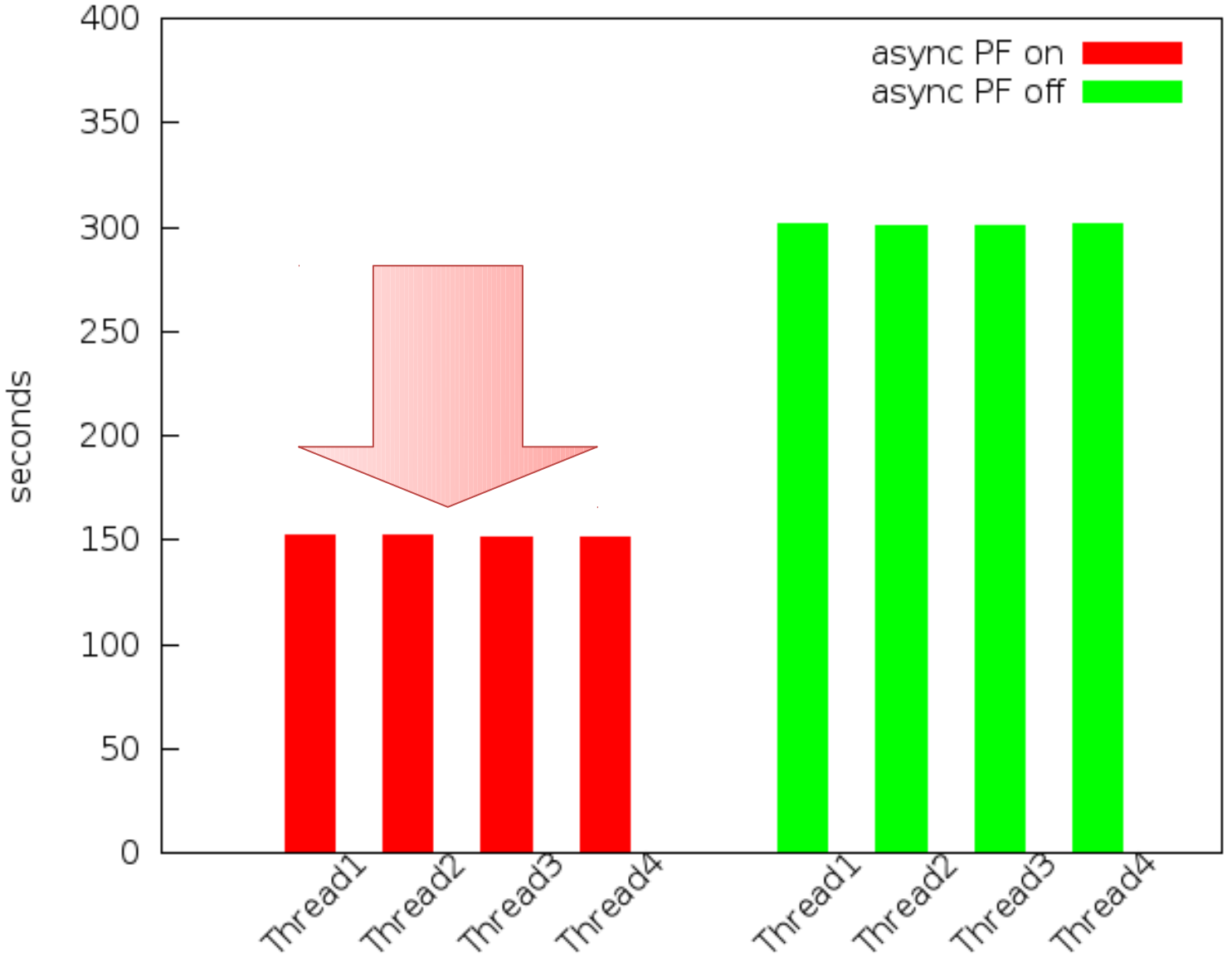
# Evaluation

# Memory scanning with postcopy

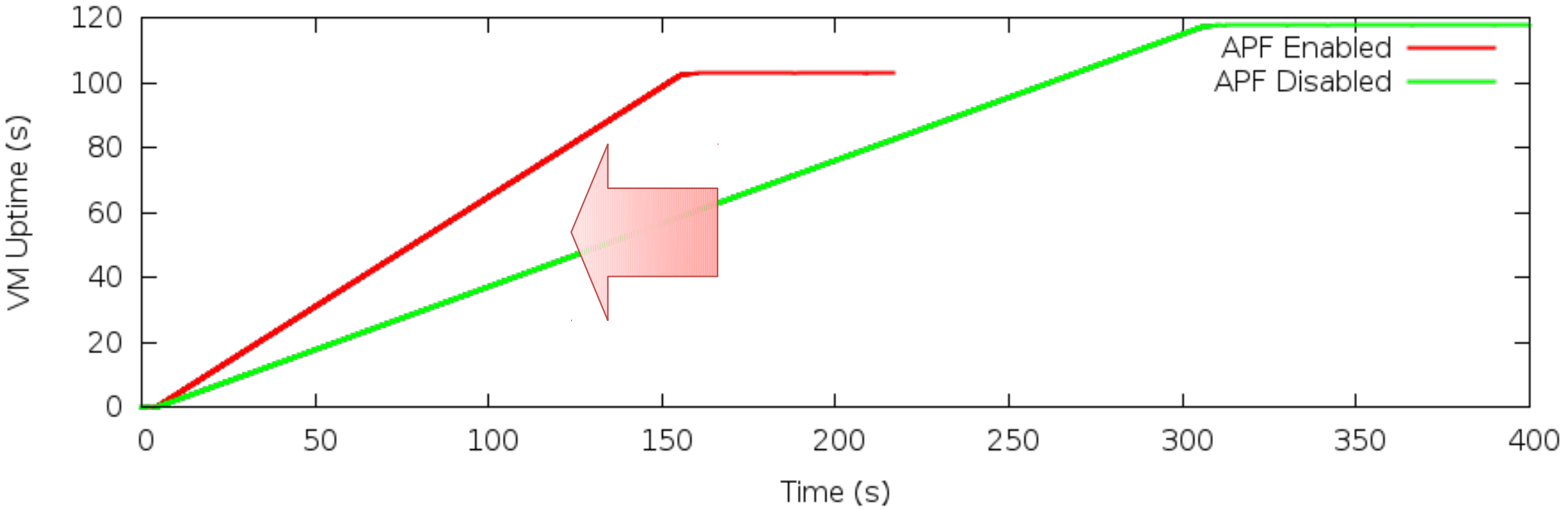
- 6GB memory Guest RAM
- 4thread
- Per-thread
  - 1GB
  - Each thread accessing all pages
  - Time from the first page access to the last page access
- Start each thread right after starting post-copy migration
- Background transfer is disabled



# Memory scan time(real)



# Total CPU time allocated to guest VM



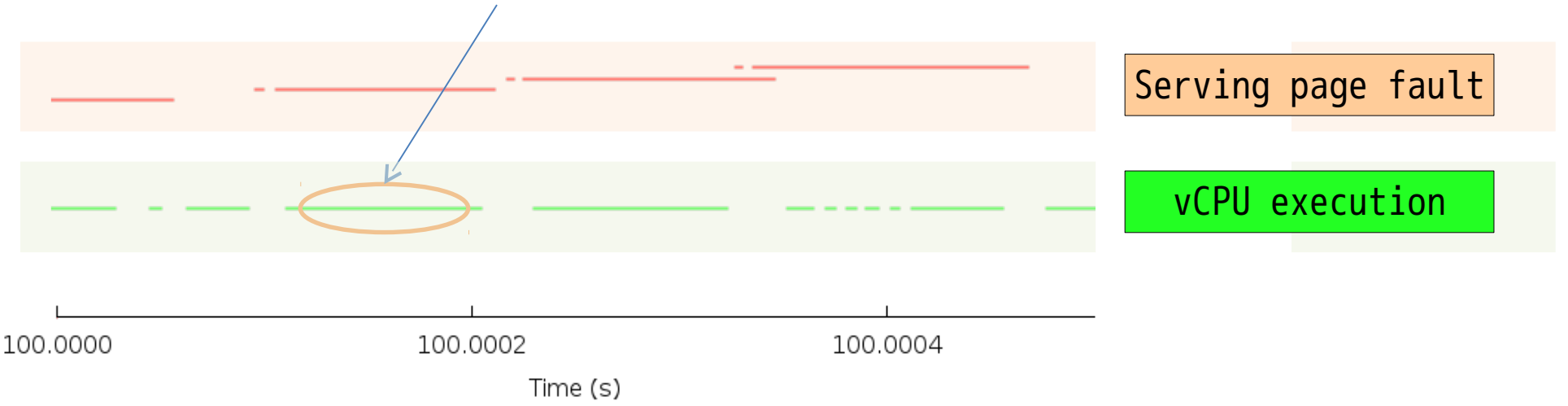

VCPU execution efficiency is improved cpu-time/real-time

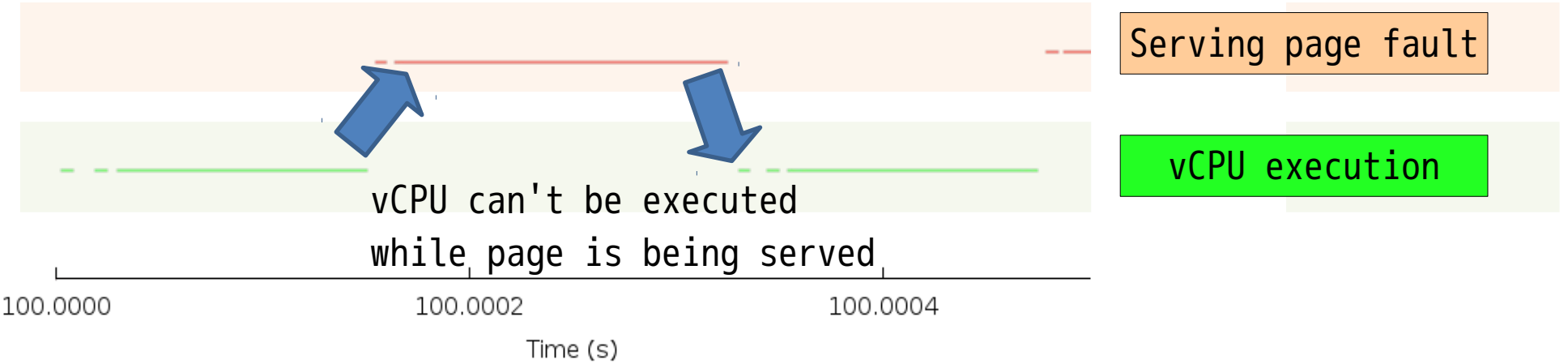
APF enabled:  0.7

APF disabled: 0.39

# Analyze with SystemTap

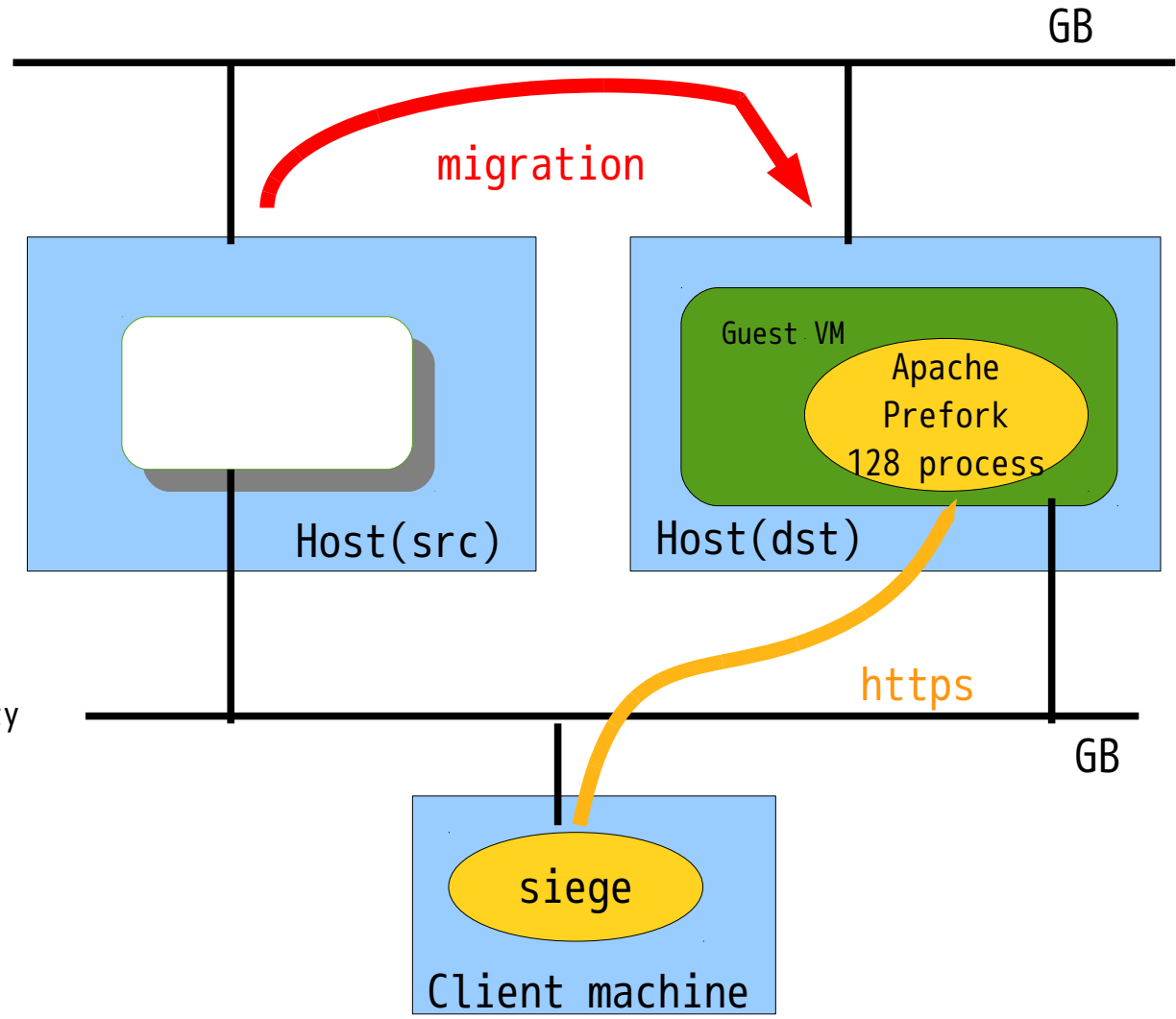
vCPU is executing during page is being served



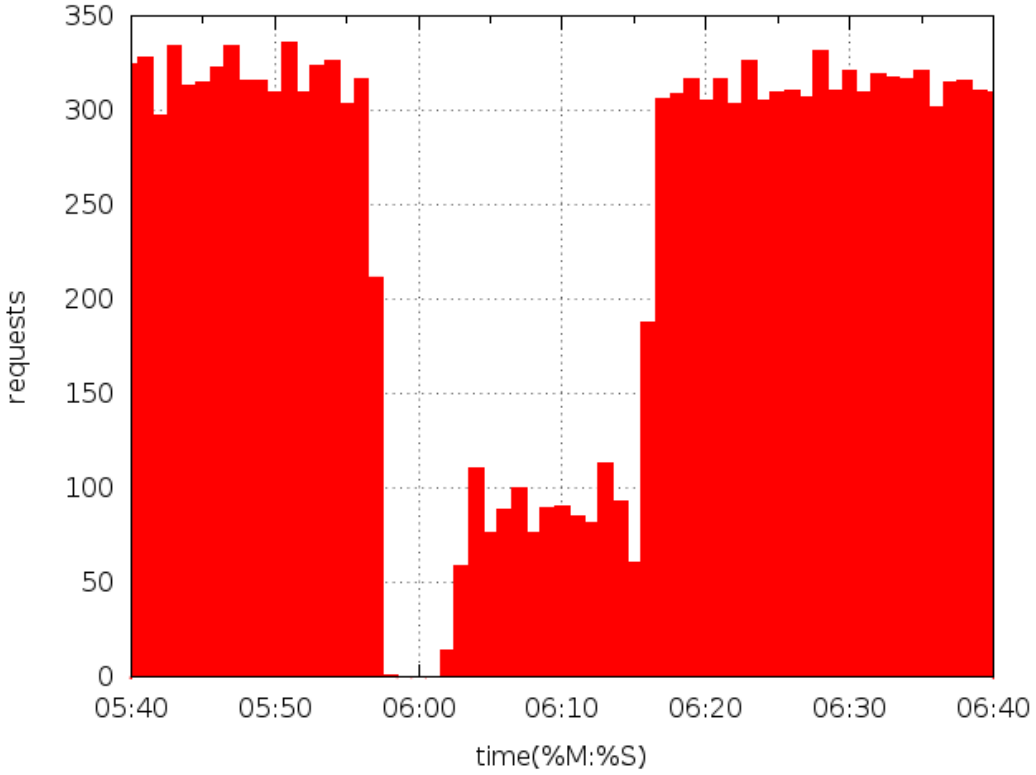



# Siege benchmark with Apache

- Host
  - Core2 quad CPU Q9400 4core
  - Memory 16GB
- Qemu/kvm
  - Vcpu = 4, kernel_irqchip=on
  - Memory about 6G(-m 6000)
  - virtio
- Apache
  - Prefork 128 process fixed
  - Data: 100K * 10000files = about 1GB
  - Warm up by siege before migration
    - So all data are cached on memory
- Siege
  - http load testing and benchmarking utility
  - http://www.joedog.org/siege-home/
  - 64 parallel (-c 64)
  - Random URL to 10000 URLs

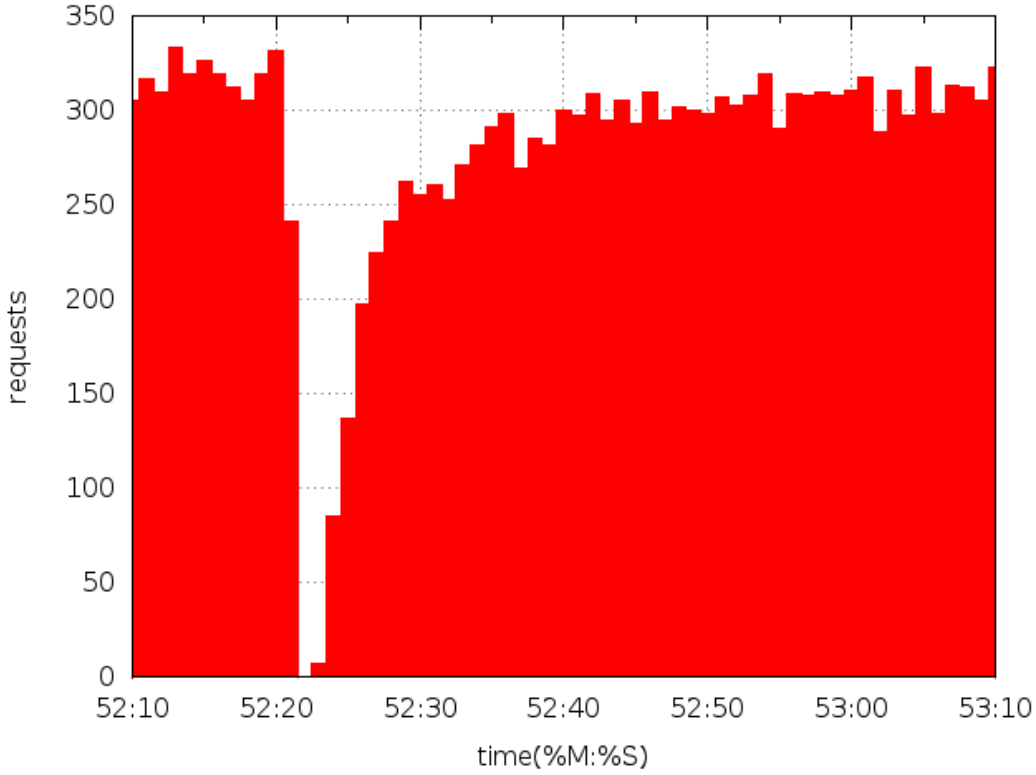


# Precopy




Precopy
Migrate set_speed=1G

# Postcopy




Postcopy w/o background transfer
Prefault forward=100
migrate -p -n tcp:10.0.0.18:4444 100 0

# Future work

- Upstream merge
  - QEmu/KVM Live-migration code needs more love
    - Code clean up, Feature negotiation...
  - Investigate for fuse version of umem device and evaluation
    - See if it's possible and its performance is acceptable
  - Evaluation/benchmark
- KSM and THP
- Threading
  - Page compression(XBRZLE) is coming. Non-blocking read + checking if all data is ready is impractical for compressed page any more.
- Mix precopy/postcopy
- Avoid memory copy
- Not to fetch pages when writing/clearing whole page
  - cleancache/frontswap might be good candidate
  - Free page aren't necessary to be transferred
    - Self-ballooning?
  - Hint via PV interface?
- Libvirt support?(and Openstack?)
- Cooperate with Kemari

# Thank you

- Questions?
- Resources
  - Project page
    - http://grivon.apgrid.org/quick-kvm-migration
    - http://sites.google.com/site/grivonhome/quick-kvm-migration
  - Enabling Instantaneous Relocation of Virtual Machines with a Lightweight VMM Extension: proof-of-concept, ad-hoc prototype. not a new design
    - http://grivon.googlecode.com/svn/pub/docs/ccgrid2010-hirofuchi-paper.pdf
    - http://grivon.googlecode.com/svn/pub/docs/ccgrid2010-hirofuchi-talk.pdf
  - Reactive consolidation of virtual machines enabled by postcopy live migration: advantage for VM consolidation
    - http://portal.acm.org/citation.cfm?id=1996125
    - http://www.emn.fr/x-info/ascola/lib/exe/fetch.php?media=internet:vtdc-postcopy.pdf
  - Qemu Wiki
    - http://wiki.qemu.org/Features/PostCopyLiveMigration
  - Demo video
    - http://www.youtube.com/watch?v=lo2JJ2KWrlA

# Backup slides

A VM is characterized by a set of memory pages $\{p_1, \ldots, p_N\}$ assumed for simplicity to be of fixed size equal to $P$ bytes. Assume the bandwidth available for the transfer is constant and equal to $b$ bytes per second (this is possible by using the techniques described in Section 4), and let $T$ denote the time interval needed to transfer a single page $T = \frac{P+H}{b}$ (under the assumption that no compression is used), where $H$ is the overhead in bytes introduced by the migration protocol for each page. Assume that, for the time horizon spanning the entire migration process, each page $p_i$ has a constant probability $\pi_i$ of being accessed at least once for writing within each time frame $T$, and assume the events of write access for each page are all independent from one another. Assume the migration process works according to the following steps:

1. at time $t_1$ in which the migration starts, the set of pages $\mathcal{D}_1$ to be transmitted is set to the entire set of pages used by the VM; let $n_1$ denote its cardinality $n_1 = |\mathcal{D}_1|$;
2. for $k = 1, \ldots, K$, repeat the following: all the $n_k$ pages in $\mathcal{D}_k$ ($n_k = |\mathcal{D}_k|$) are transferred, with a bandwidth of $b$ bytes per second, according to the order specified by the function $\phi_k : \{1 \ldots n_k\} \rightarrow \{1 \ldots N\}$ (i.e., the pages are transmitted in the order $p_{\phi_k(1)}, \ldots, p_{\phi_k(n_k)}$); the transfer ends at $t_{k+1} = t_k + n_k T$, in which $n_{k+1}$ pages $\mathcal{D}_{k+1}$ are found to have become dirty again;
3. stop the VM and transfer the last $n_{K+1}$ pages in $\mathcal{D}_{K+1}$, up to the migration finishing time $t_f = t_{K+1} + n_{K+1}\frac{P+H}{b_d}$, using a bandwidth of $b_d$ bytes per second, with $b_d \geq b$.

Then, the crucial values characterizing the migration process are the down-time $t_d = t_f - t_K$ during which the VM is stopped, and the overall migration time $t_{tot} = t_f - t_1$, which may now be expressed in terms of the other quantities introduced above:

$$t_d = \left(\frac{P+H}{b_d}\right) n_{K+1}, \quad t_{tot} = \left(\frac{P+H}{b}\right) \sum_{k=1}^{K} n_k + t_d. \tag{1}$$

The above introduced notation and assumptions are at the basis of the following results, that focus on the case $K = 1$ for the sake of brevity. All proofs are omitted but they are available at: http://retis.sssup.it/ tommaso/vhpc09-proofs.pdf.

**Proposition 1** *The probability of a page $p_i$ that is not dirty at time $t_1$ to become dirty and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1}. \tag{2}$$

**Proposition 2** *The probability of a page $p_i$ that is dirty at time $t_1$ to become dirty again and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)}, \tag{3}$$

*where $\phi_1^{-1}(\cdot) : \{1 \ldots N\} \rightarrow \{1 \ldots n_1\}$ denotes the inverse of the $\phi_1(\cdot)$ function.*

**Theorem 1** *The expected overall migration time (with $K = 1$) is:*

$$E[t_{tot}] = \left(\frac{P+H}{b}\right) n_1 + \left(\frac{P+H}{b_d}\right) \left[ n_1 - \sum_{j=1}^{n_1} \left(1 - \pi_{\phi_1(j)}\right)^{n_1+1-j} + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1 - \pi_i)^{n_1} \right]. \tag{4}$$

**Theorem 2** *The order $(\phi_k(1), \ldots, \phi_k(n_k))$ of transmission of the pages that minimizes the expected number of dirty pages found at the end of the $k^{th}$ live migration step must satisfy the following condition:*

$$\forall j \; \pi_{\phi_k(j)}(1 - \pi_{\phi_k(j)})^{n_k - j} \leq \pi_{\phi_k(j+1)}(1 - \pi_{\phi_k(j+1)})^{n_k - j}. \tag{5}$$

**Corollary 1** *If the probabilities $\pi_i$ are all lower than $\frac{1}{n_k+1}$, than the optimum ordering is obtained for increasing values of the probabilities $\pi_i$. On the other hand, if the probabilities are all greater than $\frac{1}{2}$, then the optimum ordering is obtained for decreasing values of the $\pi_i$.*