Zcache and RAMster
(oh, and frontswap too)
overview and some benchmarking

Dan Magenheimer, Oracle Corp.

Linux Storage Filesystem and Memory
Management Summit, 2012

akpm, Nov 1, 2011
Re: [GIT PULL] mm: frontswap (for 3.2 window)

(https://lkml.org/lkml/2011/11/1/317)

akpm > At kernel summit there was discussion and overall agreement that we've been paying insufficient attention to the big-picture "should we include this feature at all" issues. We resolved to look more intensely and critically at new features with a view to deciding whether their usefulness justified their maintenance burden. It seems that you're our crash-test dummy ;)

akpm > I will confess to and apologise for dropping the ball on cleancache and frontswap. I was never really able to convince myself that it met the (very vague) cost/benefit test, but nor was I able to present convincing arguments that it failed that test. So I very badly went into hiding, to wait and see what happened. What we needed all those months ago was to have the discussion we're having now.

akpm > This is a difficult discussion and a difficult decision. But it is important that we get it right. Thanks for you patience.

akpm, Nov 1, 2011
Re: [GIT PULL] mm: frontswap (for 3.2 window)

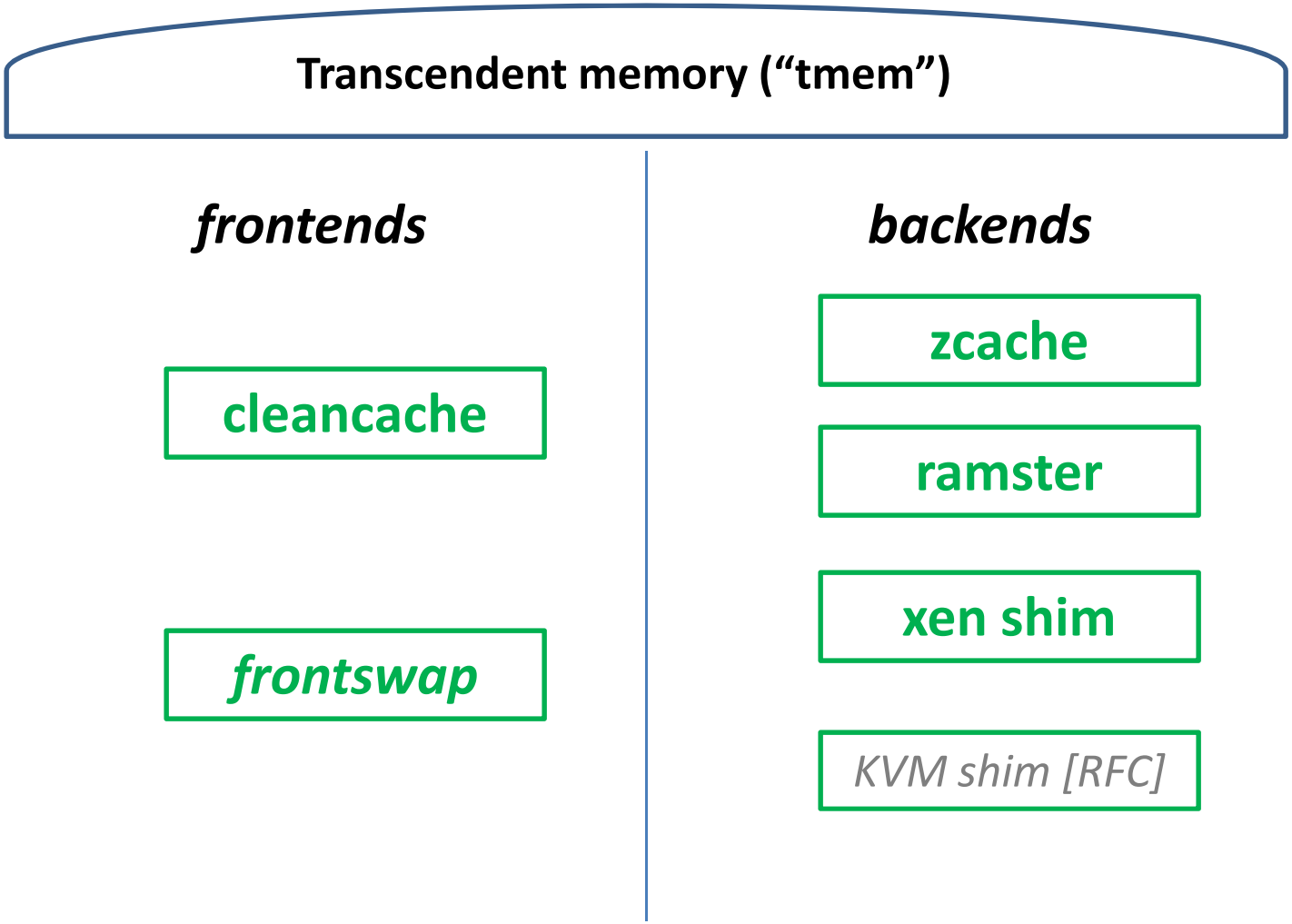
(https://lkml.org/lkml/2011/11/1/317)

> djm: OK, I will then coordinate with sfr to remove it from the linux-next
> djm: tree when (if?) akpm puts the patchset into the -mm tree.

> akpm: No, that's not necessary. The current process (you maintain git tree, it gets included in -next, later gets pulled by Linus) is good. The only reason I see for putting such code through -mm would be if there were significant interactions with other core MM work.

> akpm: It doesn't matter which route is taken, as long as the code is appropriately reviewed and tested.

Transcendent Memory and Friends



NOTE: Konrad Wilk is now the maintainer for all related parts.

Transcendent Memory "glossary"

frontends

- **data source**
- **data fetch**
- **coherency**

(clean) page cache pages — **cleancache**

swap pages — ***frontswap***

enabling hooks

backends

- **data store**
- **metadata store**

zcache — **in-kernel compression**

ramster — **cross-kernel RAM utilization**

xen shim — **spare RAM in hypervisor**

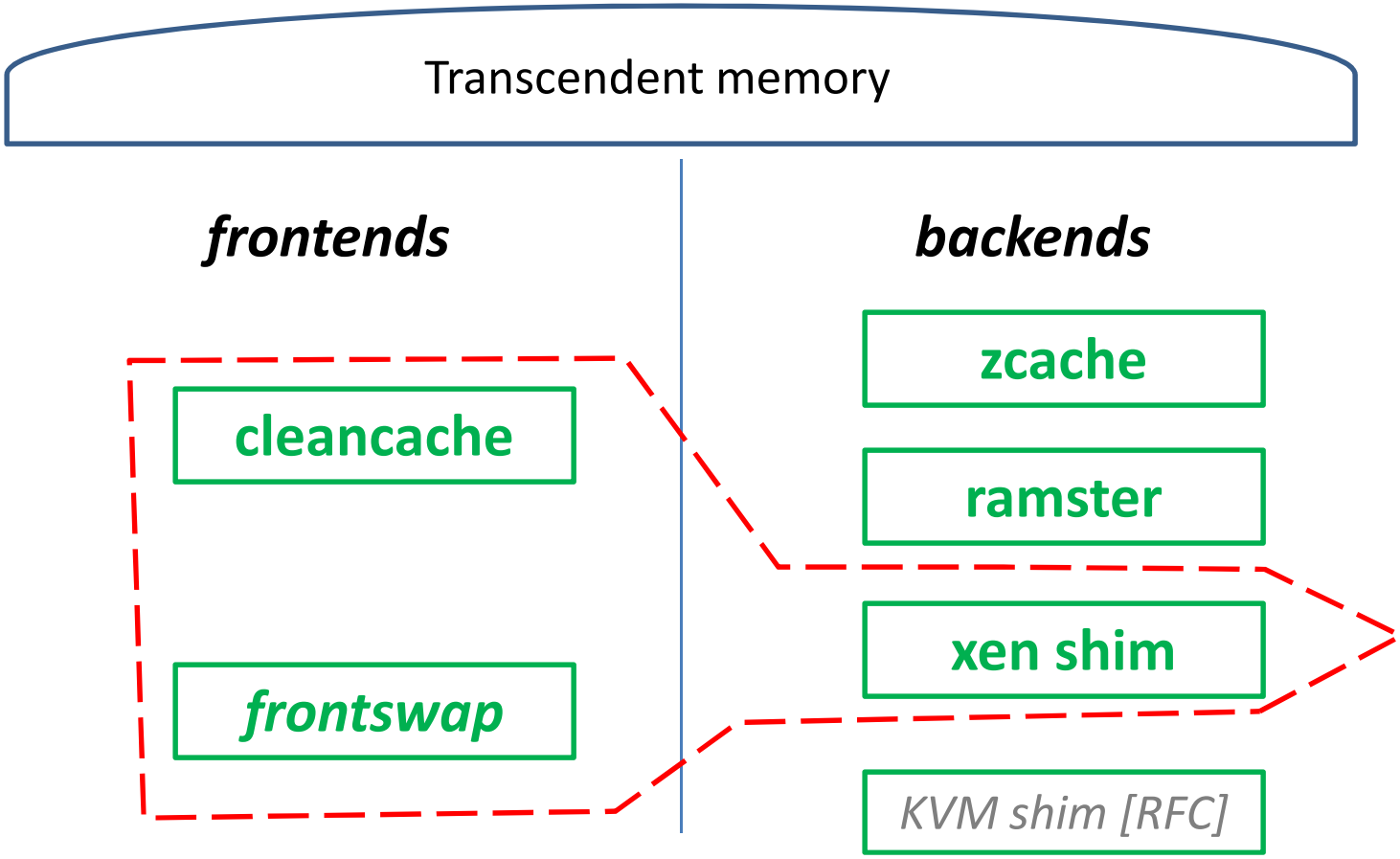
KVM shim [RFC] — use KVM host RAM

memory management value

Transcendent Memory Merge Status

frontends	*Merged at*	*backends*	*Merged at*
cleancache	**3.0**	zcache	**2.6.39 (staging)**
		ramster	**3.4 (staging)**
frontswap	***in linux-next since June 2011***	xen shim	**3.1**
		KVM shim [RFC]	

Transcendent Memory and Friends



Shipping now in Oracle UEK2 and SLES11r2

frontswap patchset diffstat

```
Documentation/vm/frontswap.txt |  210 ++++++++++++++++++++
include/linux/frontswap.h      |  126 ++++++++++++
include/linux/swap.h           |    4
include/linux/swapfile.h       |   13 +
mm/Kconfig                     |   17 ++
mm/Makefile                    |    1
mm/frontswap.c                 |  273 ++++++++++++++++++++++++++++++++
mm/page_io.c                   |   12 +
mm/swapfile.c                  |   64 +++++--
9 files changed, 707 insertions(+), 13 deletions(-)
```

- *Low core maintenance impact - ~100 lines*
- *No impact if CONFIG_FRONTSWAP=n*
- *Negligible impact if CONFIG_FRONTSWAP=y and no backend registers at runtime*
- *Then… how much benefit if a backend registers???*

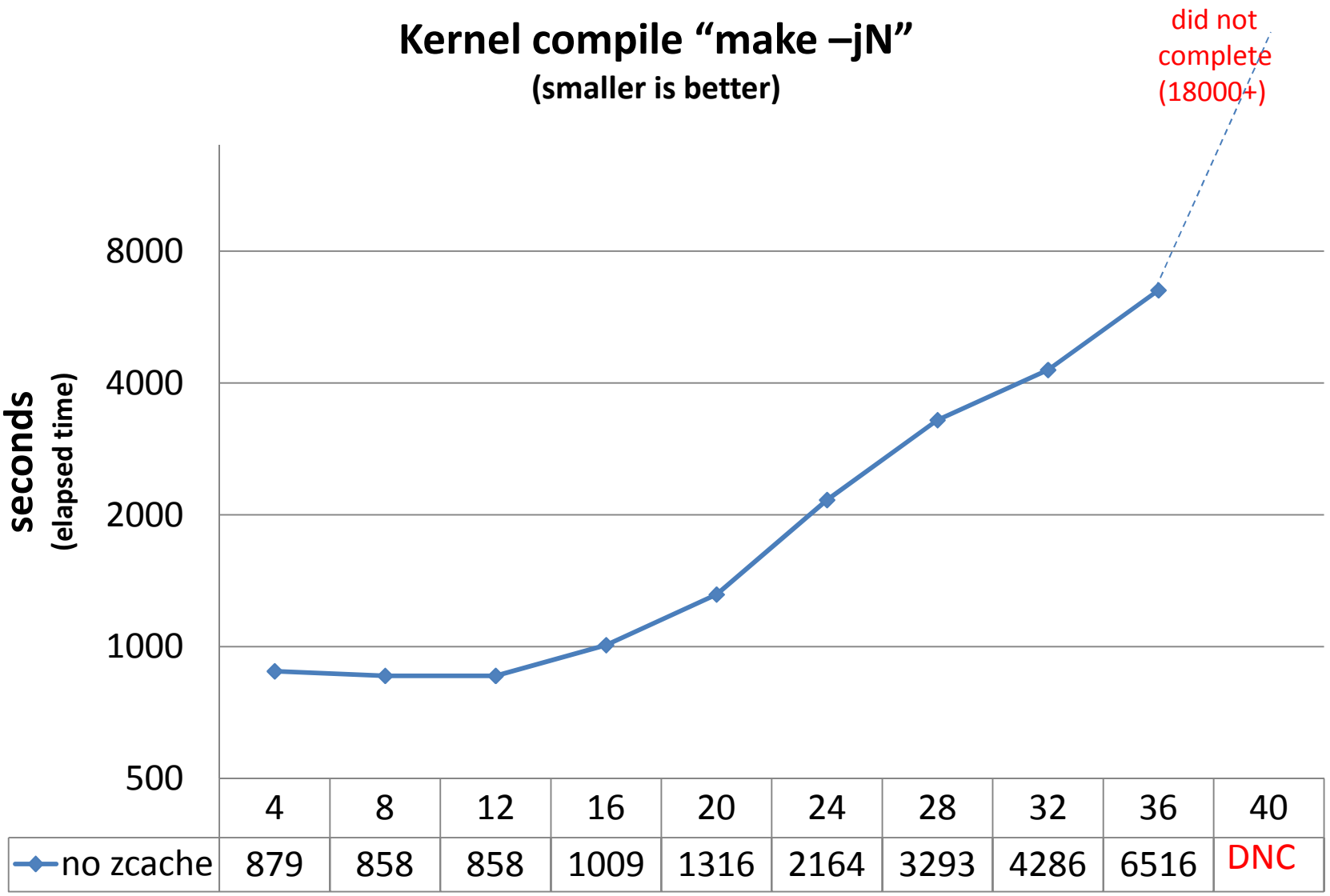
a benchmark

- Workload:
  - "make –jN" on linux-3.1 source (after make clean)
  - fresh reboot for each test run
  - all tests run as root in multi-user mode
- Software:
  - Linux-3.2
- Hardware:
  - Dell Optiplex 790 (~$500)
  - Intel Core i5-2400 @ 3.10 GHz **4core** x 2thread, 6M cache
  - ***1GB*** DDR3 DRAM 1333Mhz (limited by memmap= boot params)
  - One 7200rpm SATA 6.0Gb/s drive w/8MB cache
  - 10GB swap partition
  - 1Gb ethernet

Workload objective: Changing N varies memory pressure

- small N (4-12):
  - no memory pressure
    → page cache never fills to exceed RAM, no swapping
- medium N (16-24):
  - moderate memory pressure
    → page cache fills so lots of “reclaim”, but little or no swapping
- large N (28-36):
  - high memory pressure
    → much page cache churn, lots of swapping
- largest N (40):
  - extreme memory pressure
    → little space for page cache churn, swap storm occurs

Native (no zcache)



	4	8	12	16	20	24	28	32	36	40
no zcache	879	858	858	1009	1316	2164	3293	4286	6516	DNC

Review: what is ***zcache***?

- In drivers/staging since 2.6.39
- Captures and *compresses* evicted clean page cache pages
  - when clean pages are reclaimed (cleancache "put")
    - → zcache compresses/stores contents of evicted pages in RAM
    - → zcache has "shrinker hook" for if kernel runs low
  - when filesystem reads file pages (cleancache "get")
    - → zcache checks if it has a copy, if so decompresses/returns
    - → else reads from filesystem/disk as normal

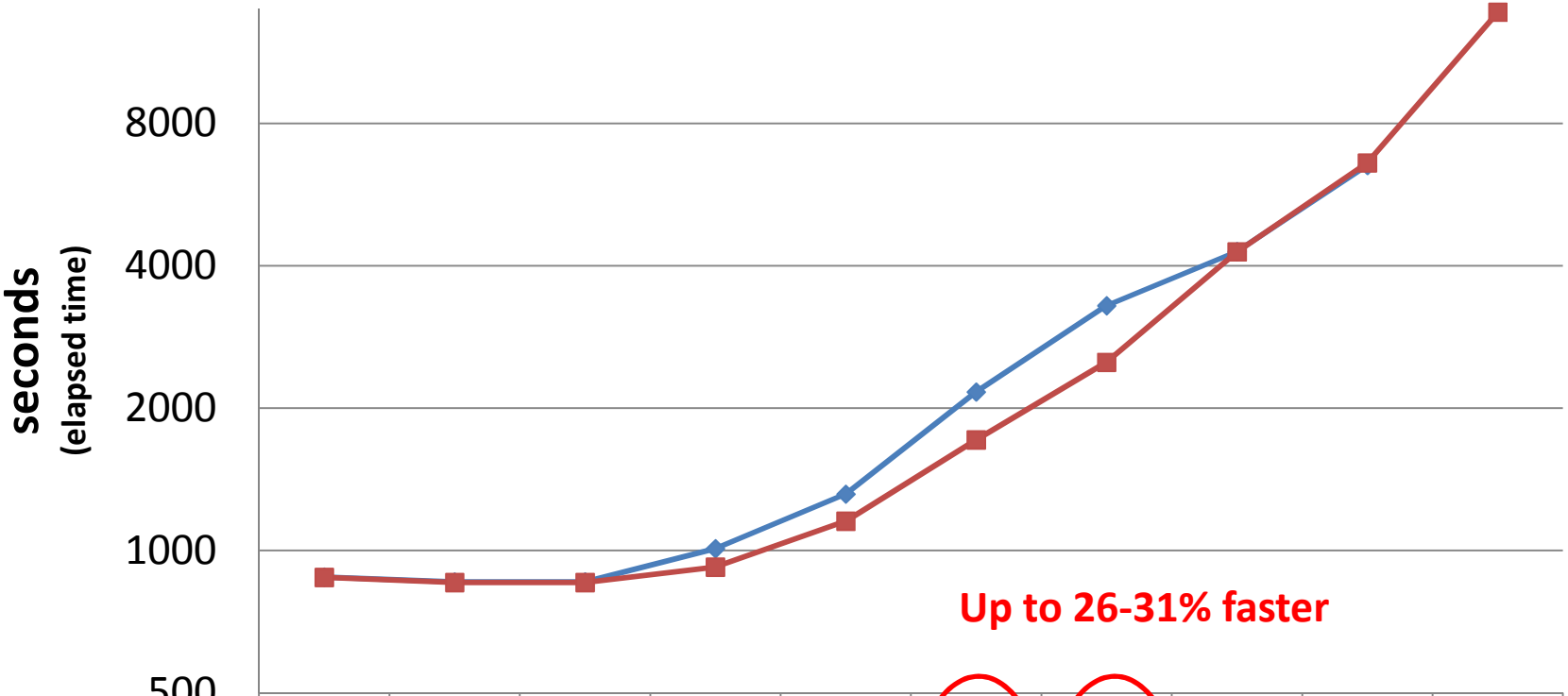
  → *One disk access saved for every successful "get"*
- Captures and compresses swap pages (in RAM)
  - when a page needs to be swapped out (frontswap "put")
    - → zcache compresses/stores contents of swap page in RAM
    - → zcache enforces policies, may reject some (or all) pages
    - → frontswap maintains a bit map for saved/rejected swap pages
  - when a page needs to be swapped in (frontswap"get")
    - → if frontswap bit is set, zcache decompresses/returns
    - →else read from swap disk as normal

  → *One disk write+read saved for every successful "get"*
- Requires cleancache hooks (merged at 3.0) and frontswap hooks (not merged yet)

zcache (vs native)

Kernel compile "make –jN"

(smaller is better)



	4	8	12	16	20	24	28	32	36	40
no zcache	879	858	858	1009	1316	2164	3293	4286	6516	
zcache	877	856	856	922	1154	1714	2500	4282	6602	13755

Benchmark analysis -- zcache

- small N (4-12):
  - no memory pressure
    - → zcache has no effect, but apparently **no measurable cost** either
- medium N (16-20):
  - moderate memory pressure
    - → zcache increases total pages cached due to compression
    - → **performance improves 9%-14% ←**
- large N (24-28)
  - high memory pressure
    - → zcache increases total pages cached due to compression
    - → AND zcache uses RAM for compressed swap to avoid swap-to-disk
    - → **performance improves 26%-31% ←**
- large N (32-36)
  - very high memory pressure
    - → compressed page cache gets reclaimed before use, no advantage
    - → compressed in-RAM swap counteracted by smaller kernel page cache?
    - → performance improves /**loses 0%-(1%)**
- largest N (40):
  - extreme memory pressure
    - → in-RAM swap compression **reduces worst case swapstorm**

Benchmark analysis – zcache (cont.)

Interesting! Why so good??

- more efficient use of RAM
  - compression benefits outweigh CPU costs
- reduced file I/O*
  - expect even more improvement in SAN
- reduced swap I/O*
  - not only avoids disk but block I/O path as well!

** see measurements posted to linux-mm by Seth Jennings (IBM)*

Review: what is ***RAMster***?

- In drivers/staging in 3.4-rc0
- Leverages zcache, adds cluster code using kernel sockets
- Locally compresses swap pages, ***but stores in remote RAM***
  - same as zcache but also "remotifies" compressed swap pages to another system's RAM

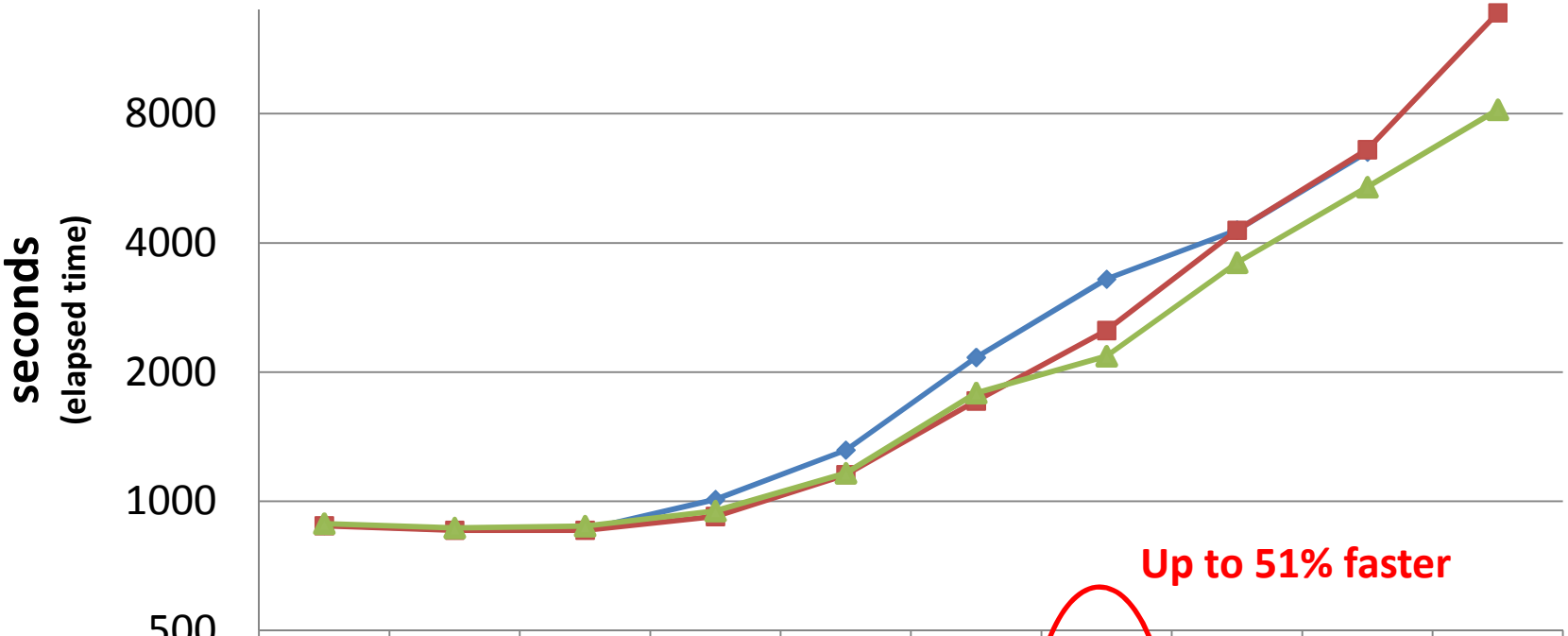
  → *One disk write+read saved for every successful "get" (at cost of some network traffic)*
- Captures and compresses clean page cache pages
  - same as zcache but optionally "remotifies" compressed pages to another system's RAM

  → *One disk access saved for every successful "get" (at cost of some network traffic)*
- Peer-to-peer or client-server (currently up to 8 nodes)
- RAM management is ***entirely dynamic***
- Requires cleancache hooks (merged at 3.0) and frontswap hooks (not merged yet); ***no other core kernel changes!***

zcache and RAMster

Kernel compile "make –jN"

(smaller is better)



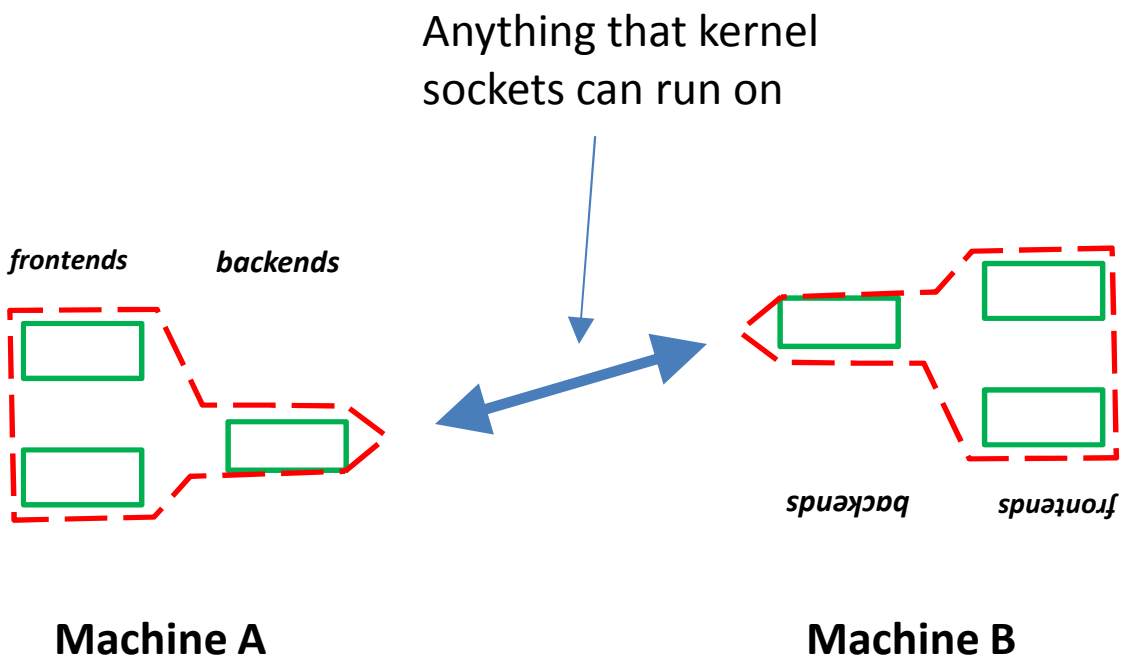
	4	8	12	16	20	24	28	32	36	40
no zcache	879	858	858	1009	1316	2164	3293	4286	6516	
zcache	877	856	856	922	1154	1714	2500	4282	6602	13755
ramster	887	866	875	949	1162	1788	2177	3599	5394	8172

Workload analysis -- RAMster

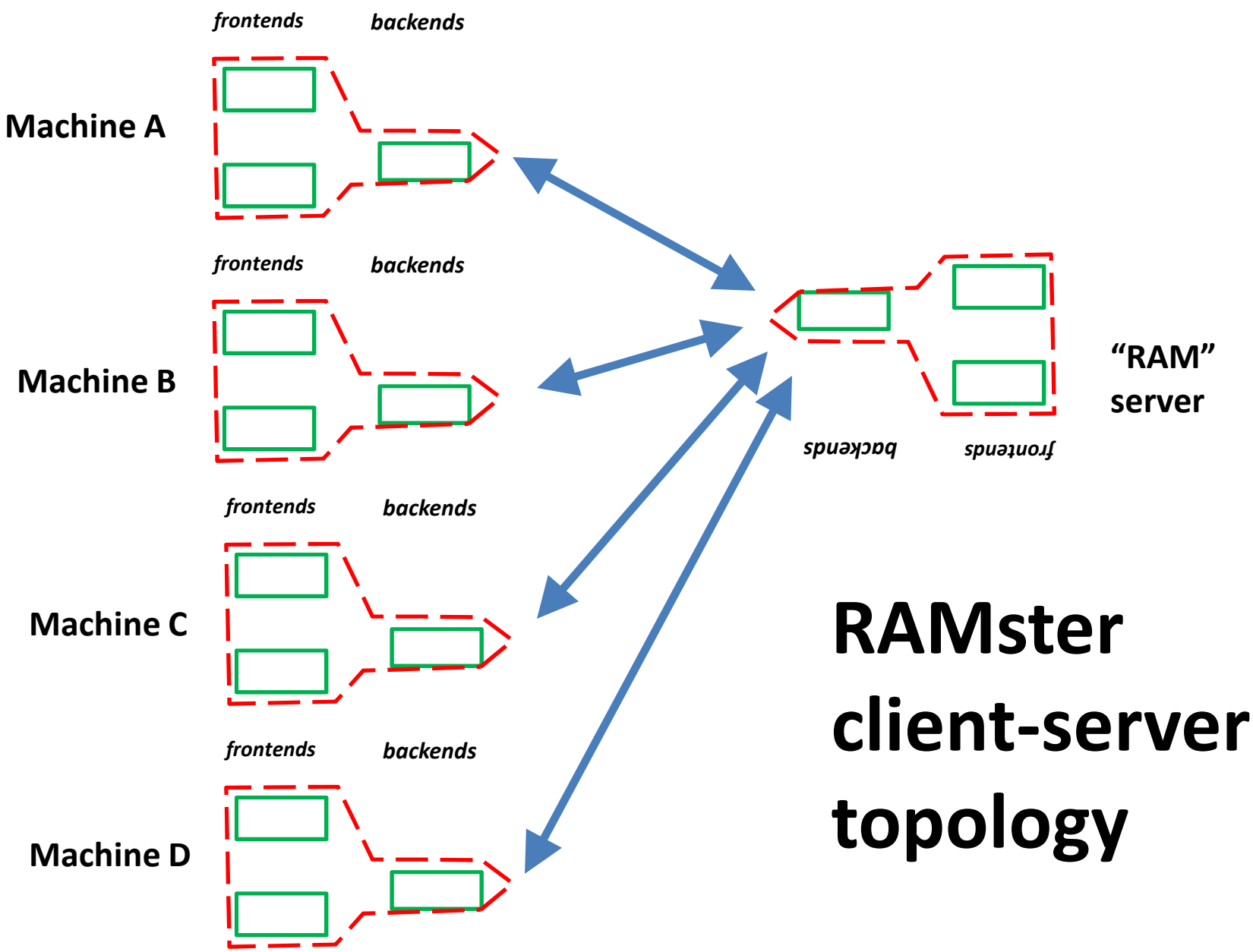
- small N (4-12):
  - no memory pressure
    → RAMster has no effect, but **small cost** *(why?)*
- medium N (16-20):
  - moderate memory pressure
    → RAMster increases total pages cached due to compression
    → **performance *improves 6%-13%***
    → **somewhat slower than zcache** *(why?)*
- large N (24-28)
  - high memory pressure
    → RAMster increases total pages cached (locall) due to compression
    → *and* RAMster uses **remote** RAM for to avoid swap-to-disk
    → **performance *improves 21%-51%***
- large N (32-36)
  - very high memory pressure
    → compressed page cache gets reclaimed before use, no advantage
    → but RAMster still uses **remote** (compressed) RAM to avoid swap-to-disk
    → **performance improves 19%-22%** (vs zcache and native)
- largest N (40):
  - extreme memory pressure
    → use of **remote** RAM **significantly reduces worst case swapstorm**

Input requested

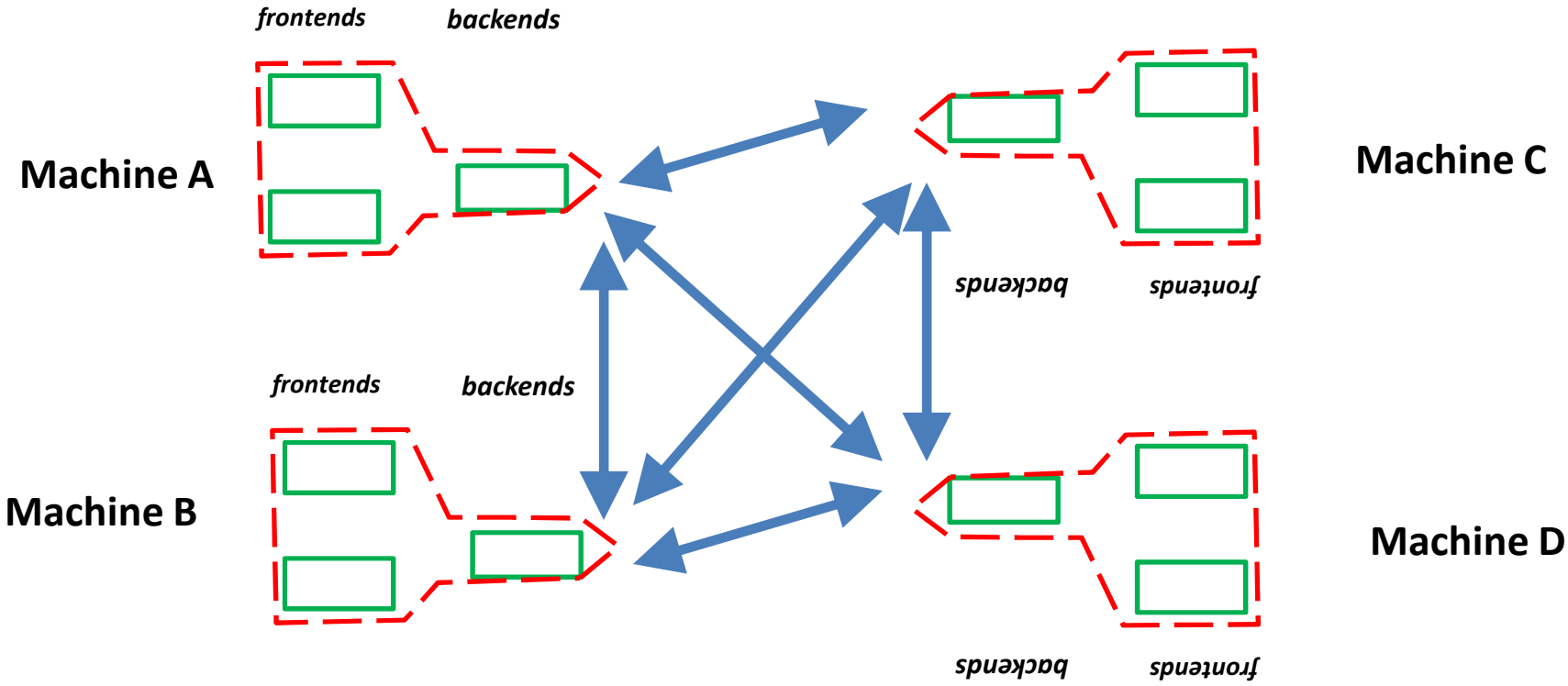
- Sufficient evidence of value to merge?
  - frontswap
  - zcache *(promotion from staging)*
    - *suggestions for where this code should live?*
      (e.g. mm/, mm/zcache/*, mm/tmem/*, drivers/zcache/* …)*
    - RAMster is logically a superset of zcache but has a few more challenges due to clustering, so plan to deal with it separately
- If not, suggestions for specific next steps to demonstrate value to merge later?



RAMster peer-to-peer



RAMster client-server topology



RAMster highly-available ("RAID"?) topology *(future)*