IO-less Dirty Throttling

Fengguang Wu
<fengguang.wu@intel.com>

LINUXCON JAPAN 2012

intel®

Open Source Technology Center

Write and writeback

USER APPLICATION

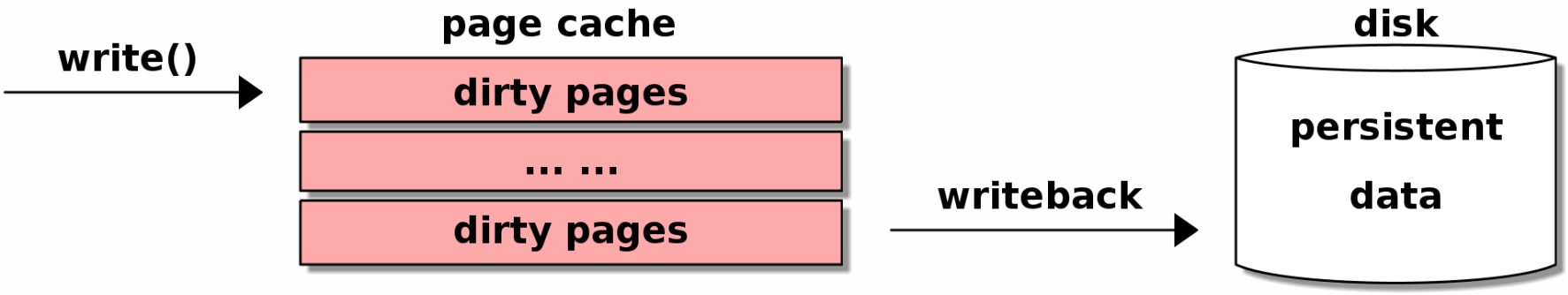
==========

dirty pages and immediately return

KERNEL FLUSHER THREAD

==========

writeout pages after dirty expired (>30s)



avoid blocking apps

aggregate write IO

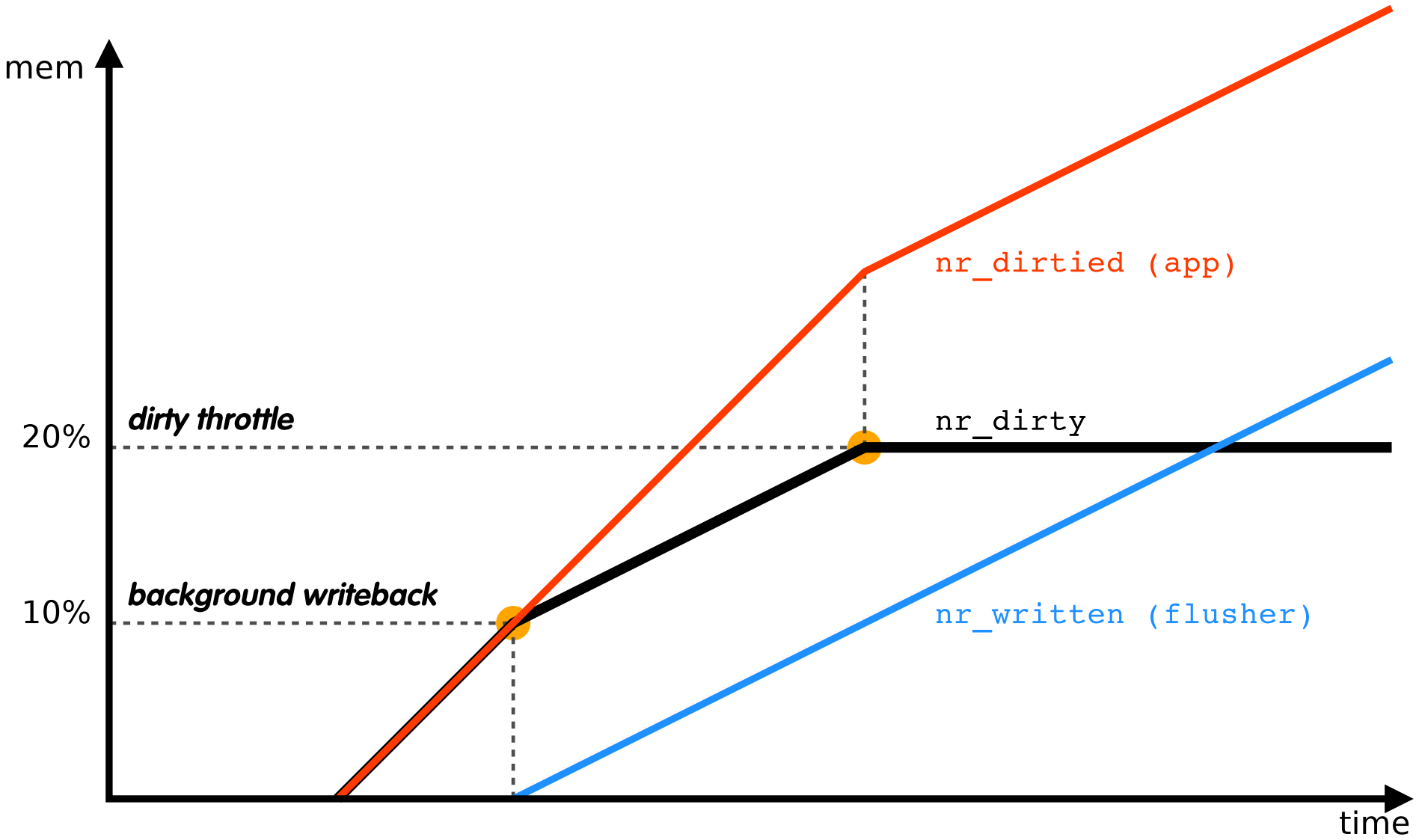
The flusher thread(s)

- **Initiate writeback IO in background**
- **One flusher per storage device**

```
$ ps ax
   PID TTY      STAT   TIME COMMAND
  2322 ?        S      0:01 [flush-8:0]
 12681 ?        S      0:00 [flush-btrfs-1]
```

Dirty throttling

To limit dirty pages.. by slowing down heavy dirtiers.

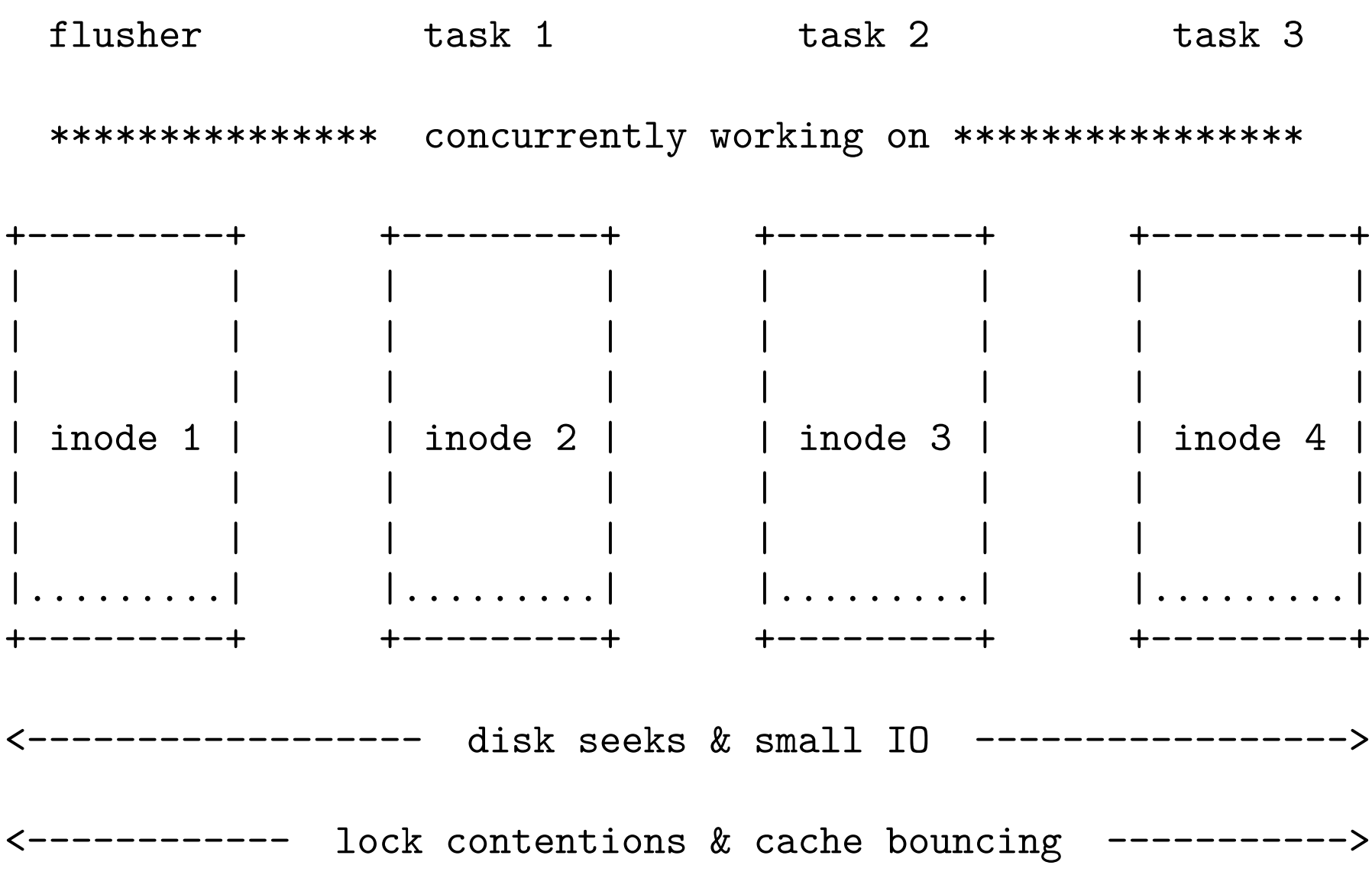


IO-full balance_dirty_pages()

```
sys_write()
    balance_dirty_pages()
```

```
        if (over_bground_thresh())
                start_background_writeback();

        if (dirty exceeded)
                writeback_inodes(pages_dirtied * 3/2);
```

Problem: disk seeks + lock contentions



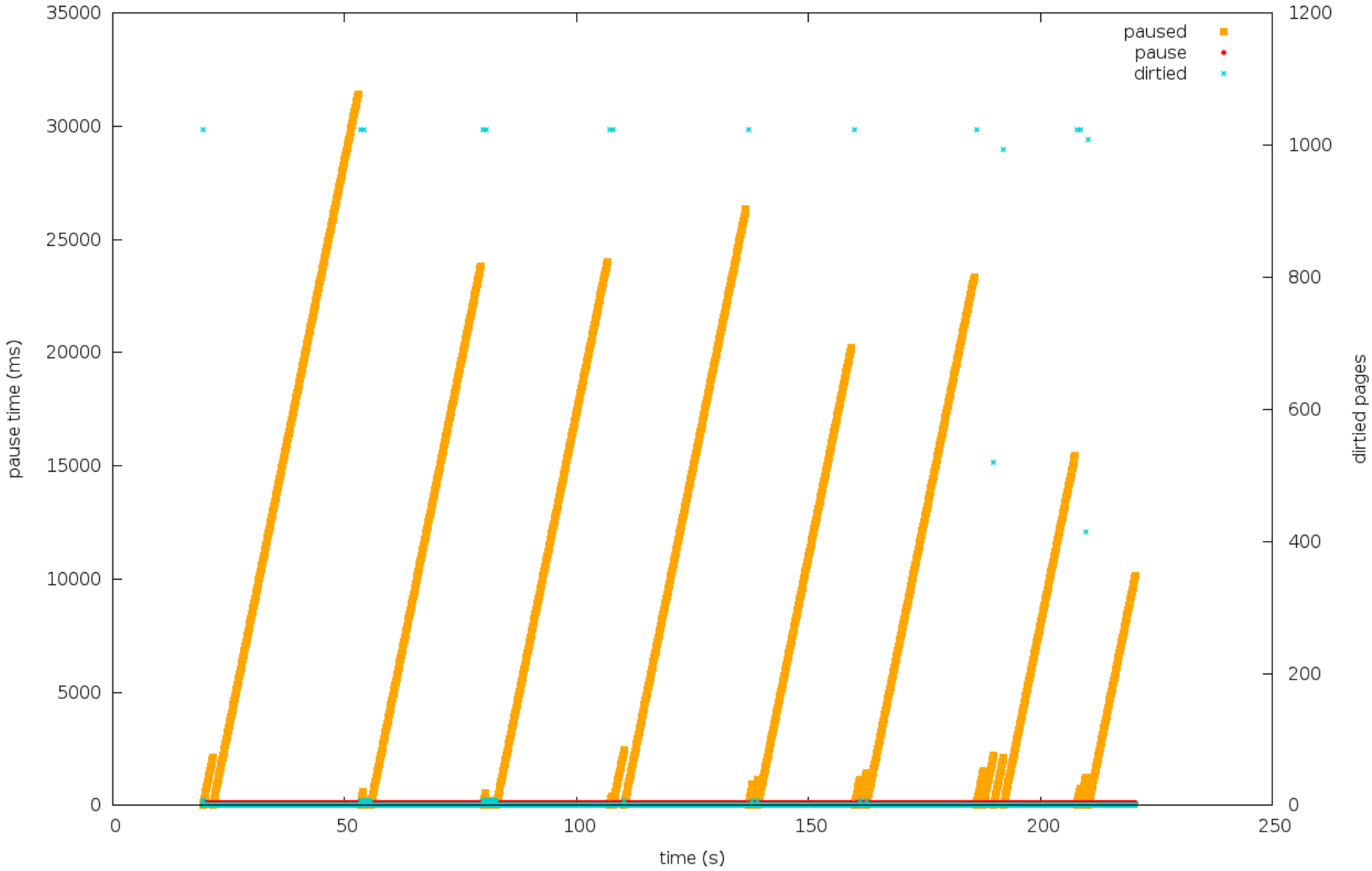
Problem: large latencies

- **tasks in deep IO path are *not killable***
- **hurts *responsiveness***
- **hurts *throughput* (pipeline stalls)**

```
loop {
        READ from net
        WRITE to disk  # long sleeps => idle network
}
```

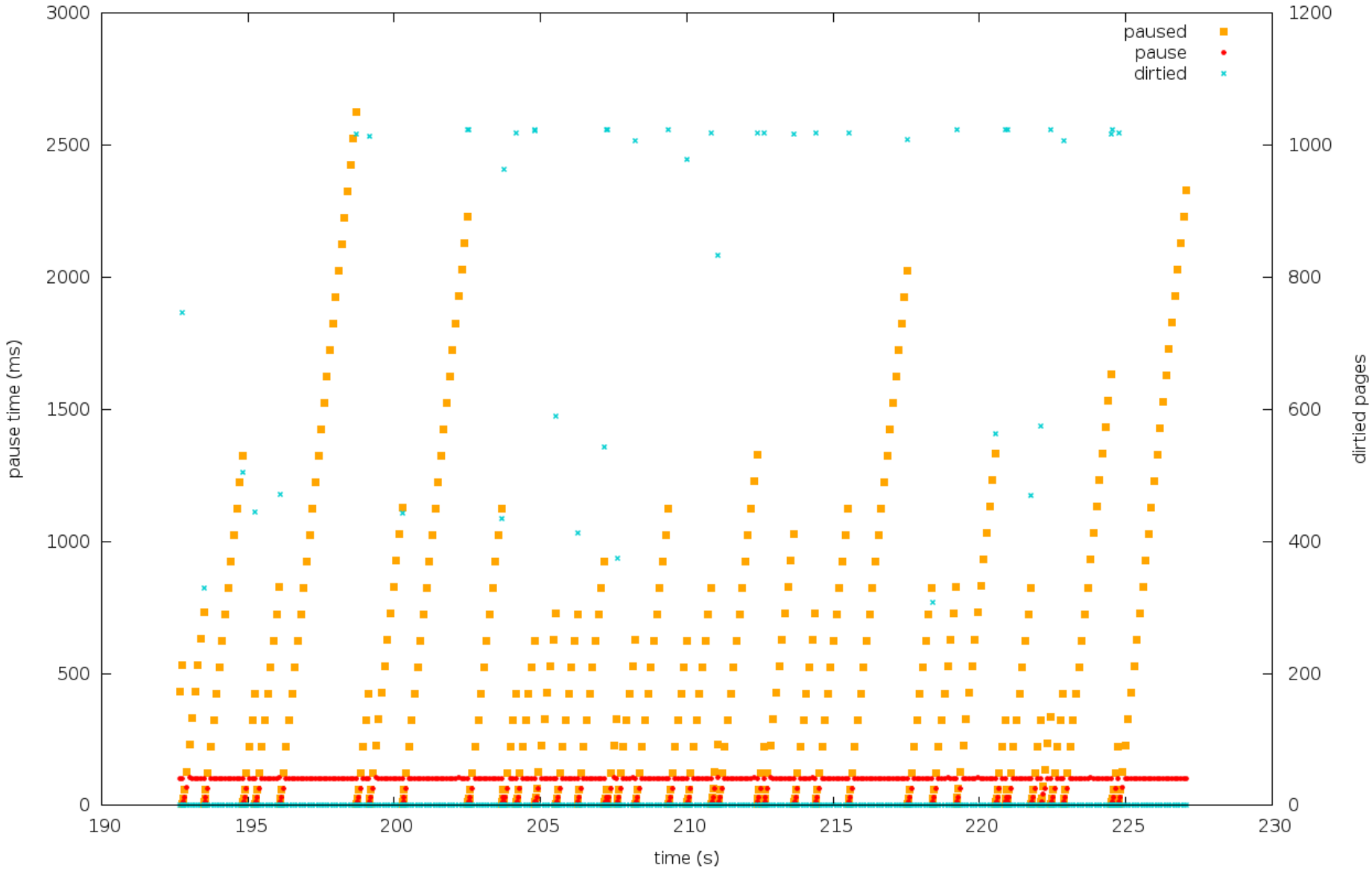
```
rsync to a loaded server:
        pause time:     0-3s     0-300ms    60-70ms
        throughput:        1      +12.4%     +63.5%
```

30s pause (NFS, 1-dd)



NFS, 1 dd, 3G mem, 2.6.38-rc7

3s pause (xfs, 8-dd)



xfs, 8 dd, 4G mem, 2.6.38-rc7

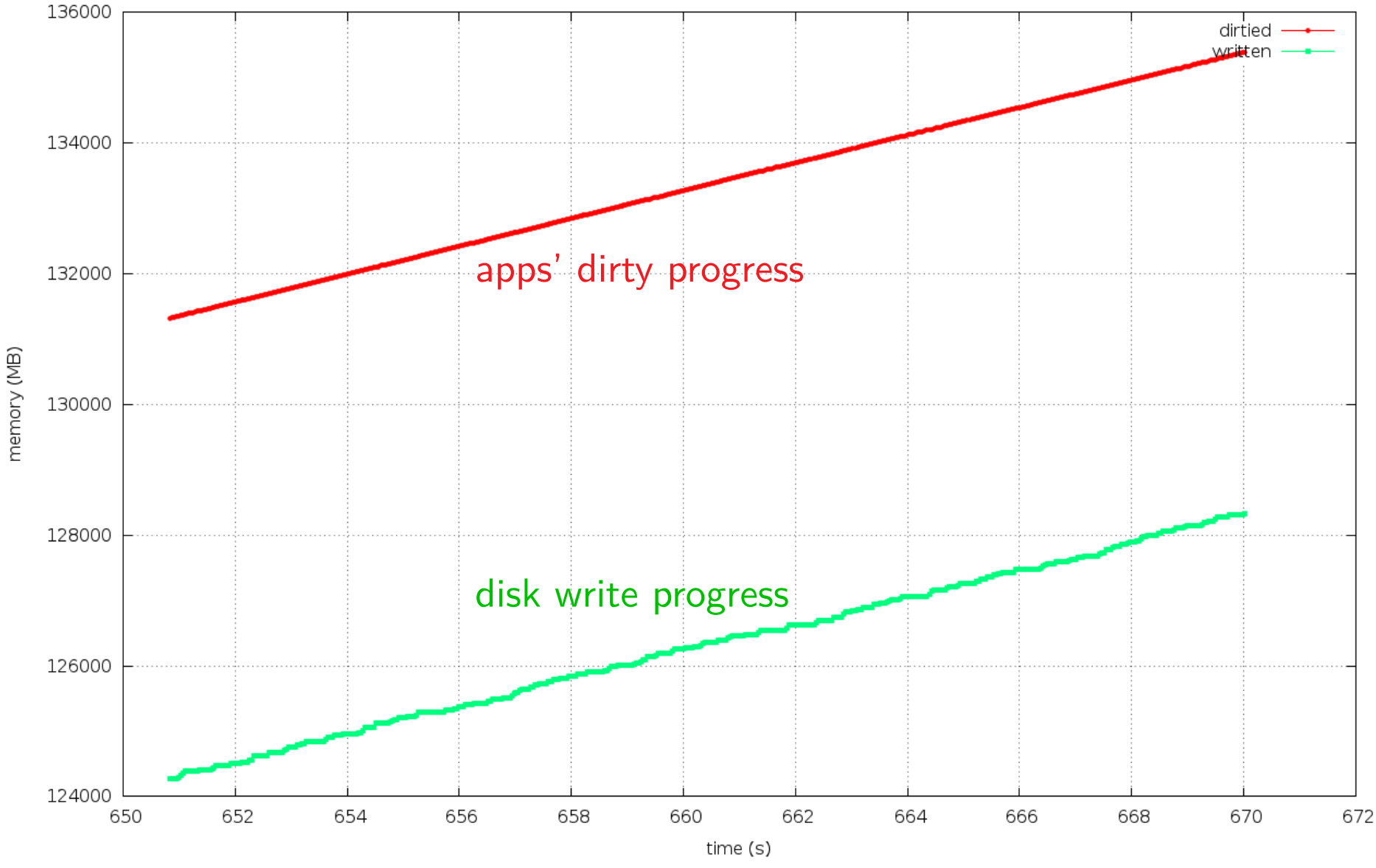
Let's do controllable sleeps!

I/O $\Rightarrow$ SLEEP

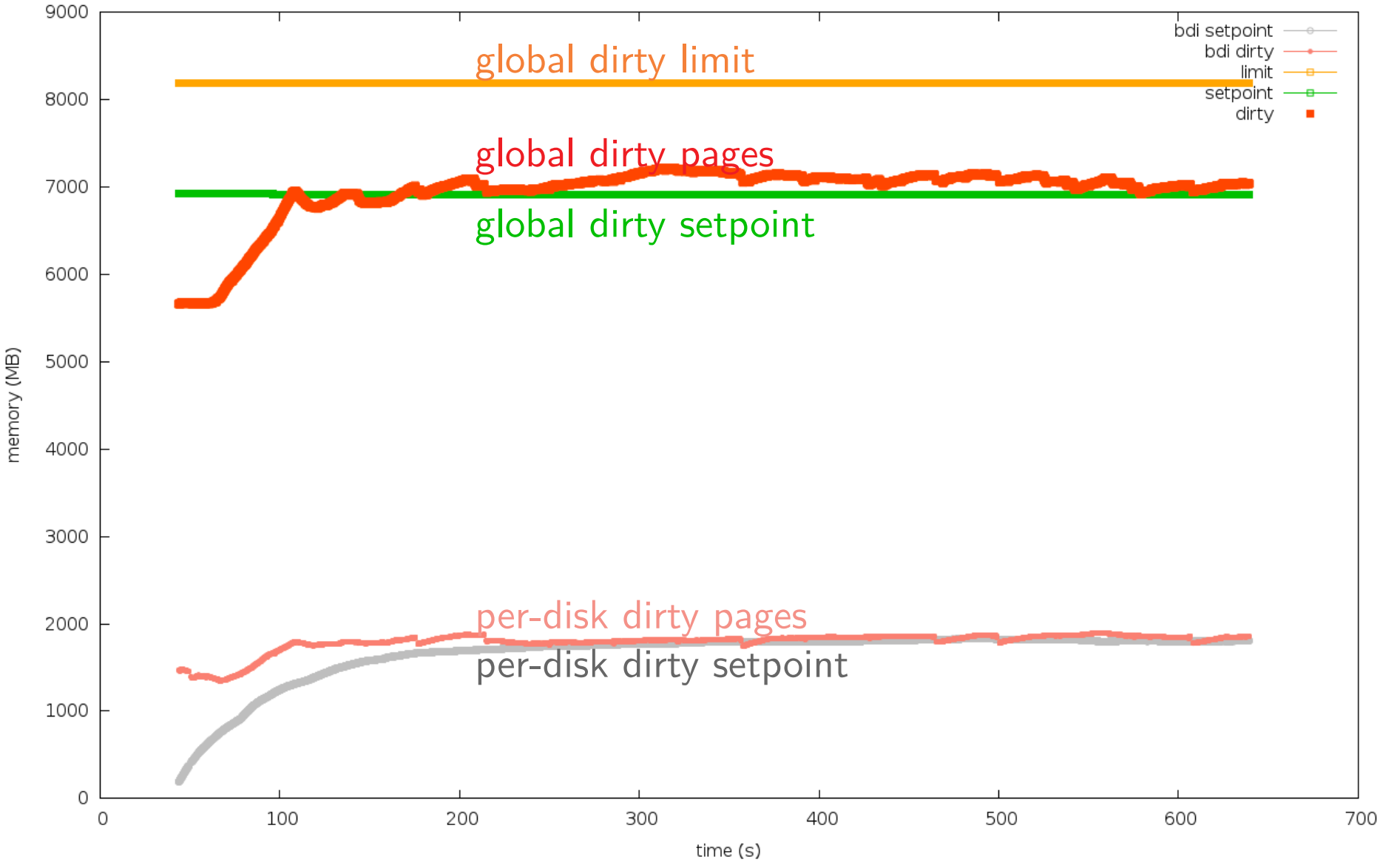
```
        if (dirty exceeded)
-               writeback_inodes(pages_dirtied * 3/2);
+               sleep(pages_dirtied / task_ratelimit);
```

and rethink dirty balancing ……

Balancing dirty rate



Balancing dirty pages

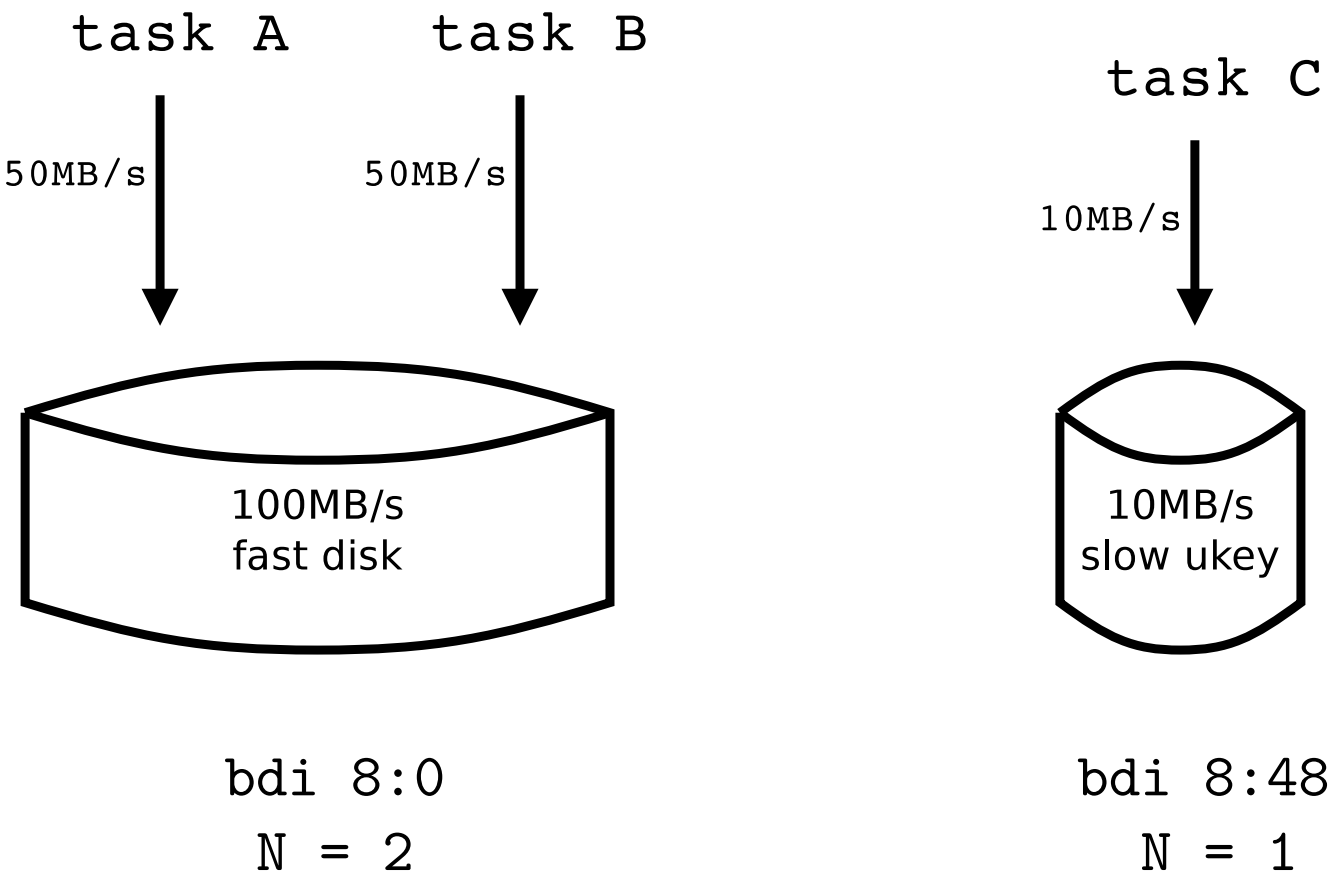


IO-less dirty throttling/balancing

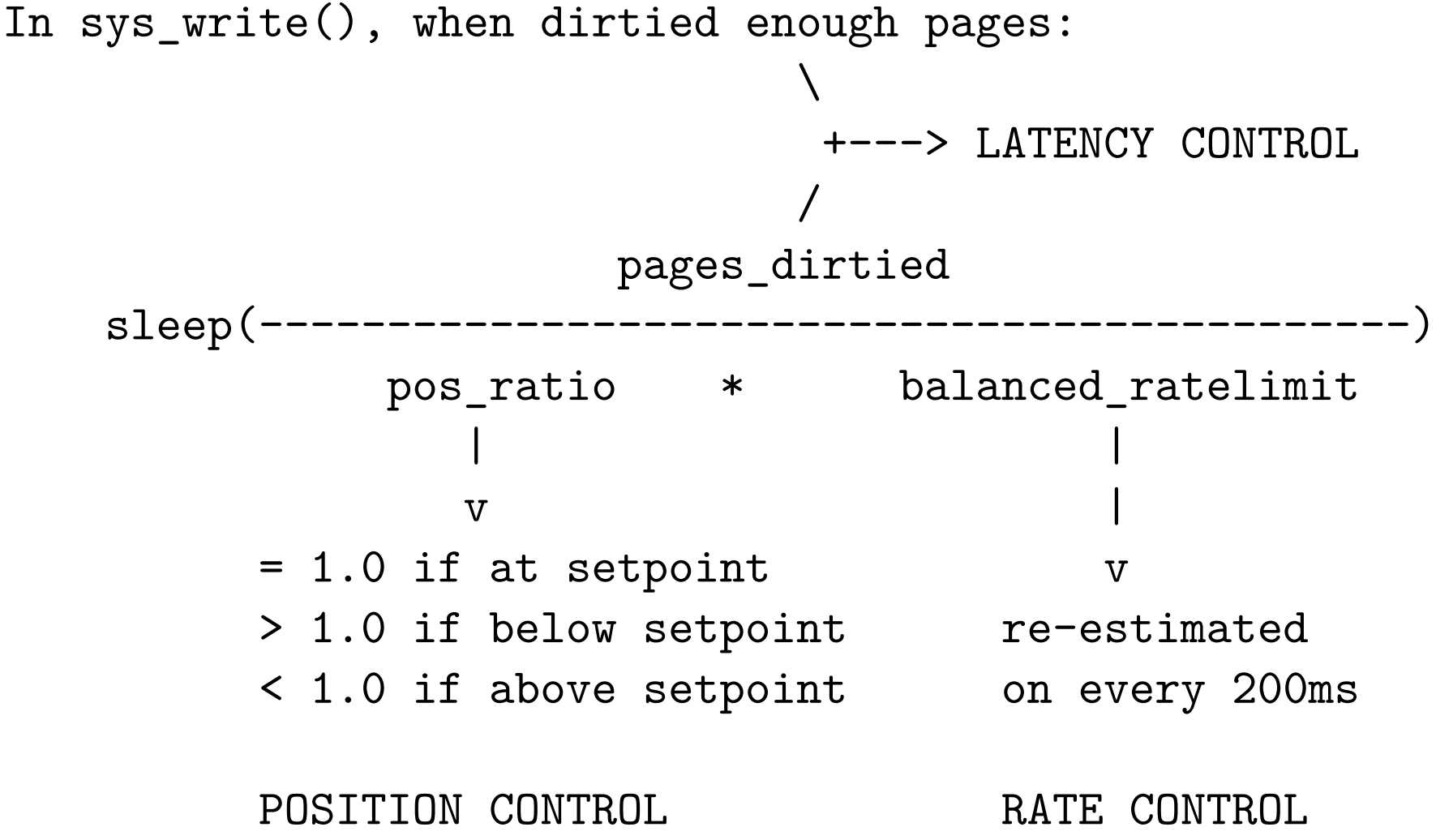
Do proper `sleep()`s during `sys_write()`:

1. **throttle each task's *dirty rate* around `write_bandwidth / N`**
2. **balance the global *dirty pages* around global dirty target**
3. **balance each disk's *dirty pages* around the disk's dirty target**

per-disk write bandwidth shares



The 3 axes of sleeping



Controllable `sys_write()` latency

```
        if (enough pages dirtied)
                sleep(pages_dirtied / TASK_RATELIMIT);
```

Controlling `pages_dirtied` controls pause time.

Reasonable `sys_write()` latencies

100MB/s => insert 10-seconds-long sleeps into 1GB writes

```
10 * sleep(1000ms)      # too long latency

100 * sleep(100ms)
1000 * sleep(10ms)

10000 * sleep(1ms)      # too much CPU overheads
```

Pause time policies

- **10ms for 1-dd**
- **more dirtier tasks $\Rightarrow$ longer pause time**
- **200ms max pause**

Regressions on tiny `pages_dirtied`

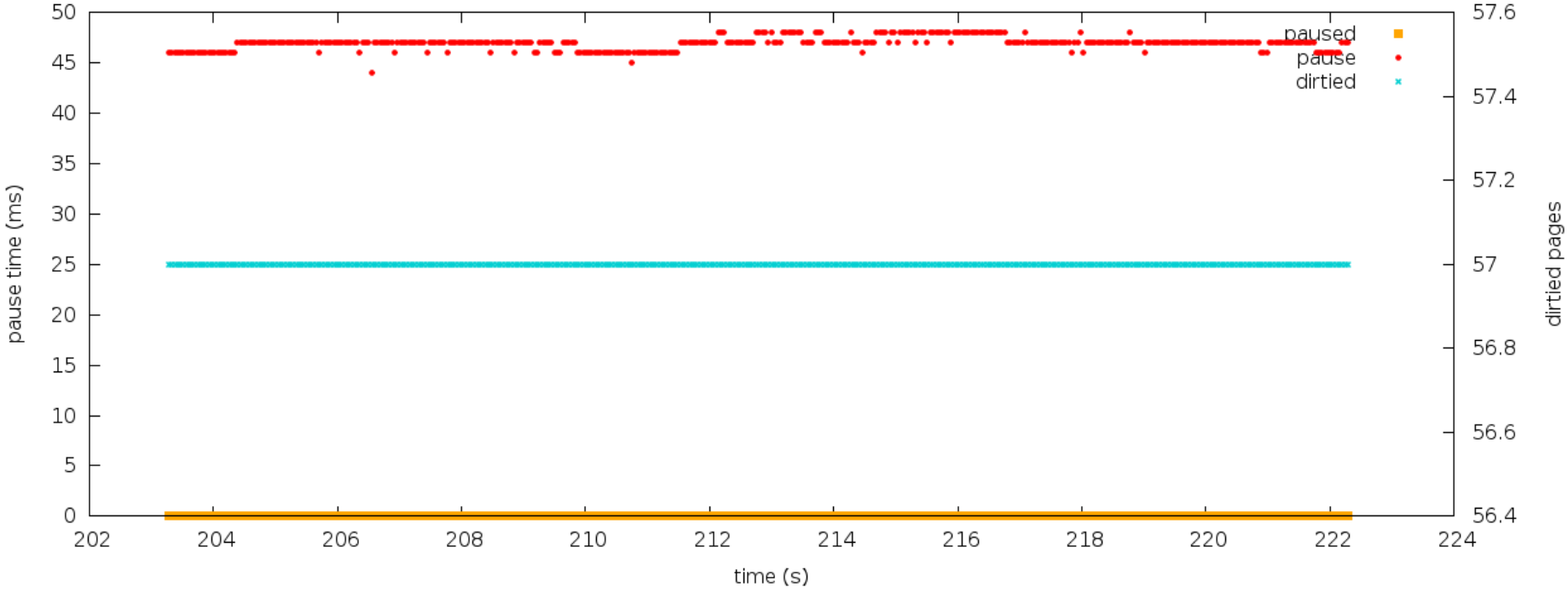
WORKLOAD $64kb$ random writes and reads+writes

PROBLEM paused on every *few* pages dirtied

SOLUTION *increase* pause time to *increase* `pages_dirtied`

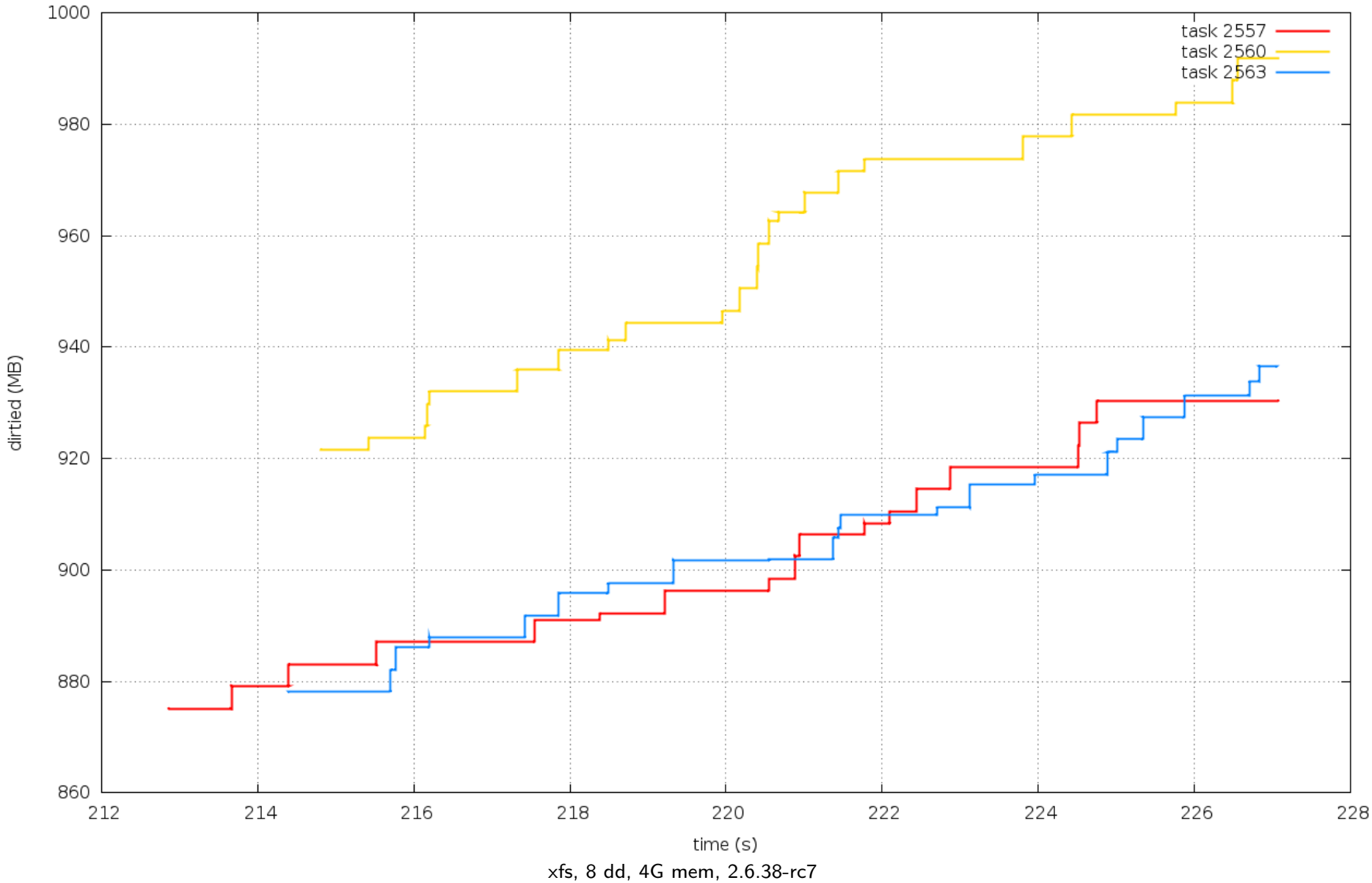
This task: pauses 45-50ms on every 57 pages dirtied

- `balance_dirty_pages()` **do sleeps in a** `for(;;)` **loop**
- `pause` **is the pause time in current loop**
- `paused` **is the accumulated pause times in previous loops**



xfs, 10 dd, 4G mem, 2.6.39-rc2-dt7+

Smoothness: bumping ahead (legacy kernel)

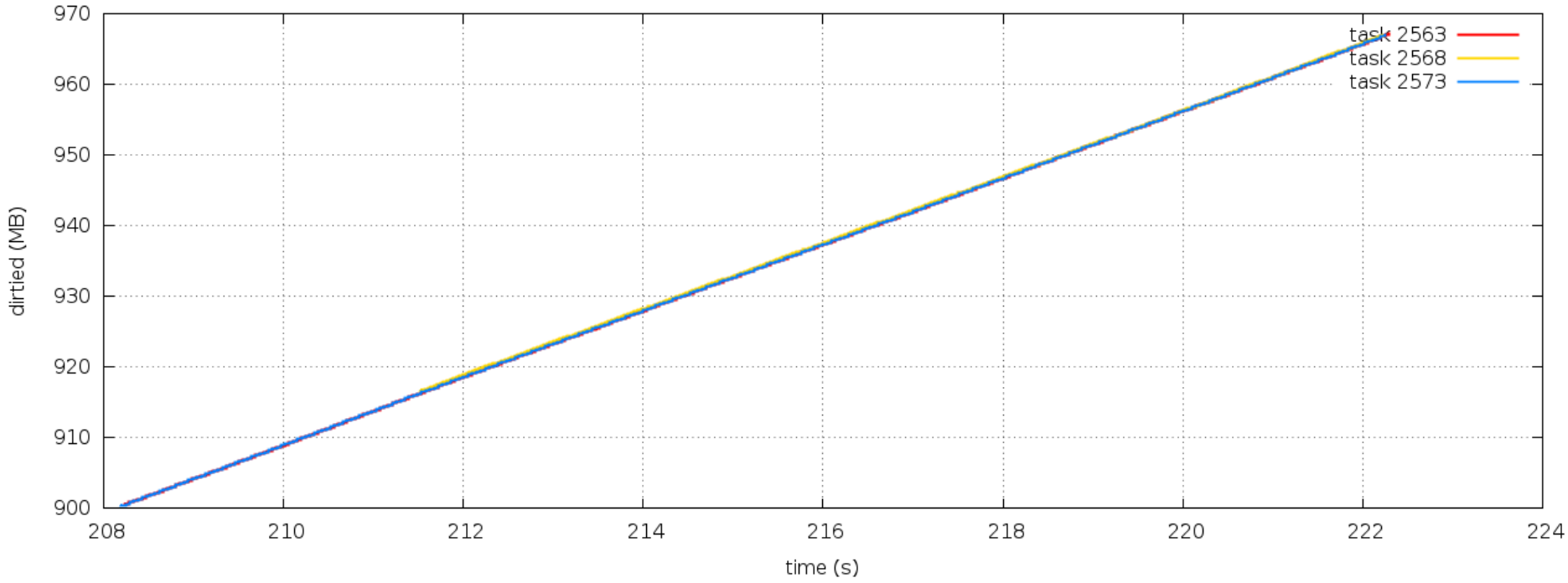


xfs, 8 dd, 4G mem, 2.6.38-rc7

Smoothness: straight lines

3 superposed lines

$\Rightarrow$ excellent smoothness and fairness among the 3 dd tasks



xfs, 10 dd, 4G mem, 2.6.39-rc2-dt7+

Balanced ratelimit

```
sleep(pages_dirtied / balanced_ratelimit);
```

That throttles the task's dirty rate to

$$\texttt{balanced\_ratelimit} = \texttt{write\_bandwidth} / \texttt{N},$$

where

$$\texttt{N} := \text{NUMBER OF DIRTIER TASKS}$$

is to be estimated.

Estimating N (theory)

When started N dd, throttle each dd at

ratelimit_0 (any non-zero initial value is OK)

After 200ms, we measures

dirty_rate = # of pages dirtied by apps / 200ms

to estimate N:

N = dirty_rate / ratelimit_0

Now the tasks can be rightfully throttled at

balanced_ratelimit = write_bandwidth / N

...

`balanced_ratelimit` estimated in 200ms!

However, real world is not perfect . . .

Unstable balanced_ratelimit

ext4, 1 dd to HDD + 1 dd to USB stick, 3G mem, 3.2.0-rc3

Balanced dirty ratelimit (practice)

COMPRESS SMALL NOISES

balanced_ratelimit is fluctuating, has estimation errors due to control lags and write_bw errors, asking for step-by-step approximations:

```
dirty_ratelimit => balanced_ratelimit (the closer, the smaller step)
```

FILTER OUT LARGE NOISES

There is no need to update dirty_ratelimit during a stable workload, which only makes it susceptible to (big) noises. So do it defensively and update dirty_ratelimit when balanced_ratelimit and task_ratelimit are at the same direction of dirty_ratelimit.

```
if (dirty_ratelimit > both balanced_ratelimit, task_ratelimit)
    dirty_ratelimit => max(balanced_ratelimit, task_ratelimit)

if (dirty_ratelimit < both balanced_ratelimit, task_ratelimit)
    dirty_ratelimit => min(balanced_ratelimit, task_ratelimit)
```

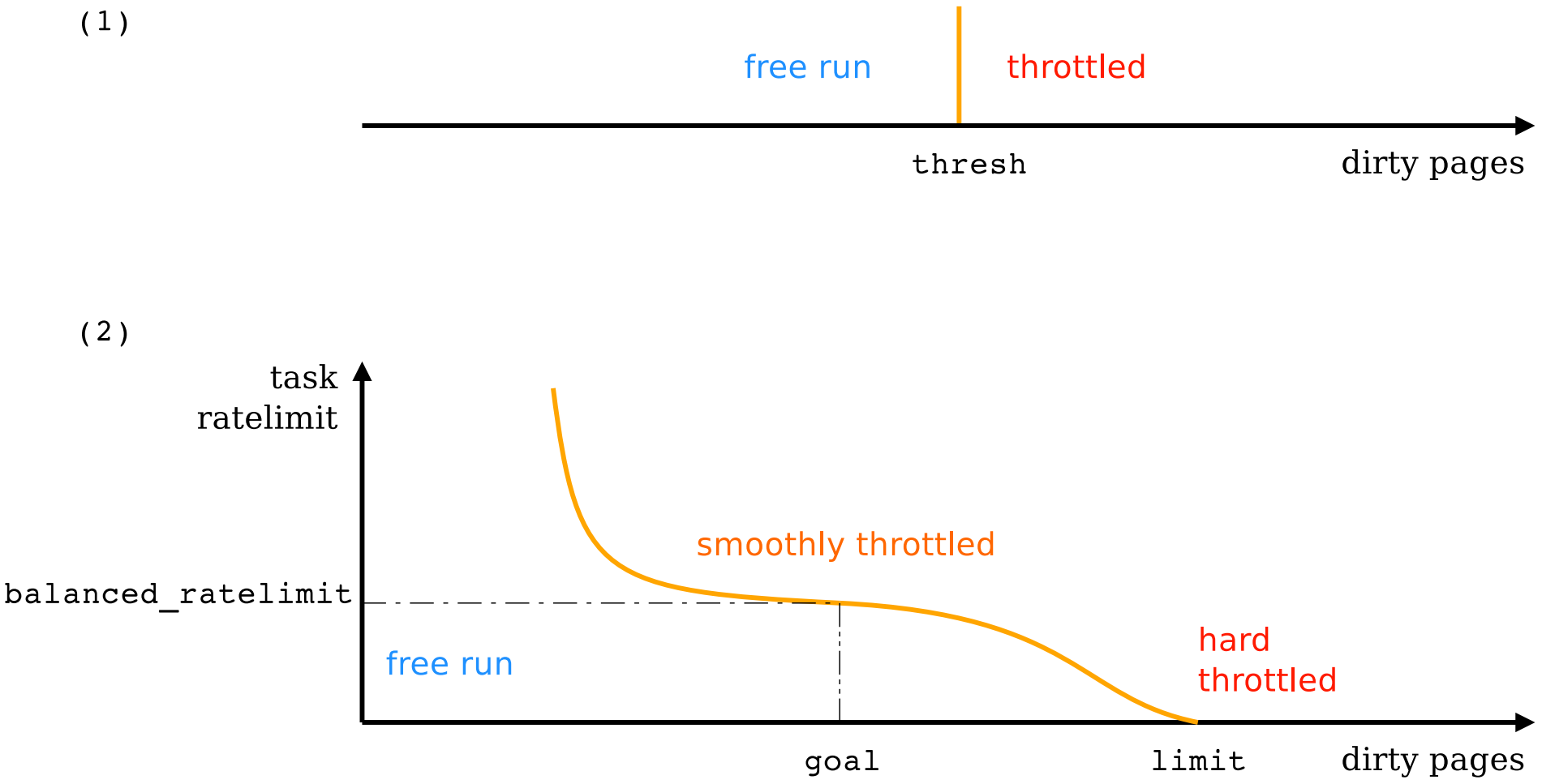
Balanced dirty pages

```
-       task_ratelimit = balanced_ratelimit;
+       task_ratelimit = balanced_ratelimit * pos_ratio;
```

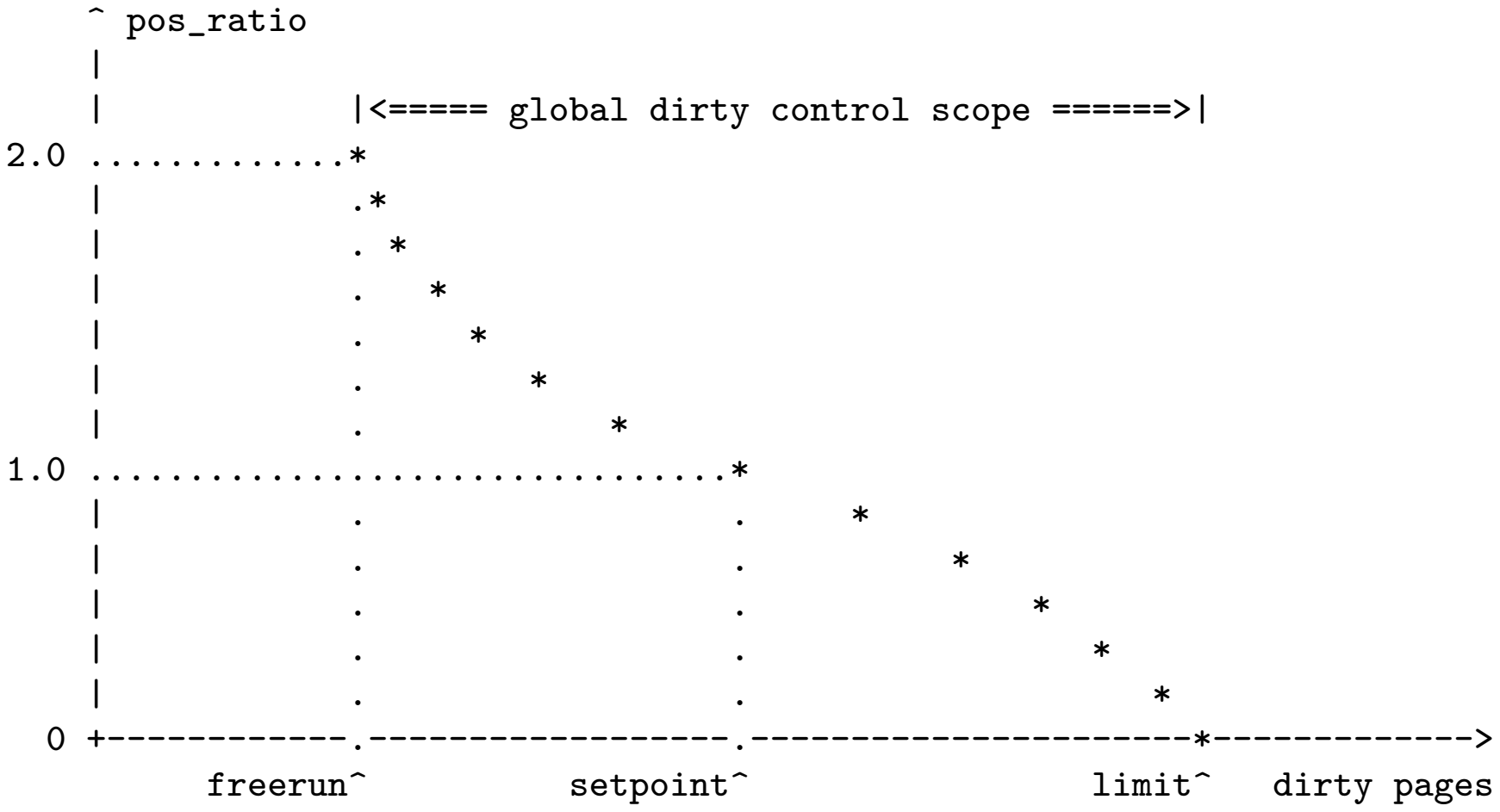
Negative feedback control

```
pos_ratio = 1.0

if (dirty < setpoint) scale up   pos_ratio
if (dirty > setpoint) scale down pos_ratio

if (bdi_dirty < bdi_setpoint) scale up   pos_ratio
if (bdi_dirty > bdi_setpoint) scale down pos_ratio
```

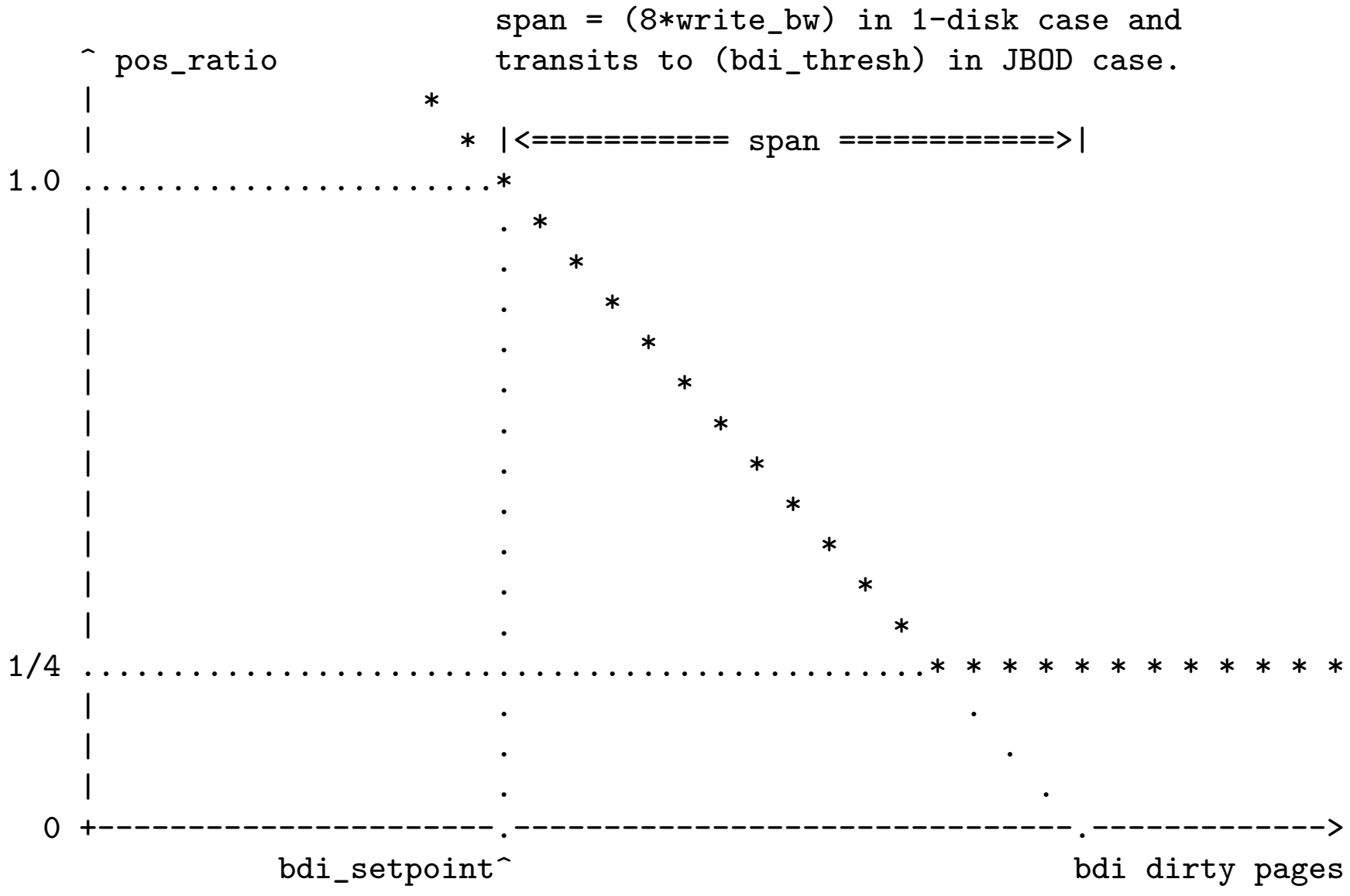
Paradigm change



Global control line



bdi control line

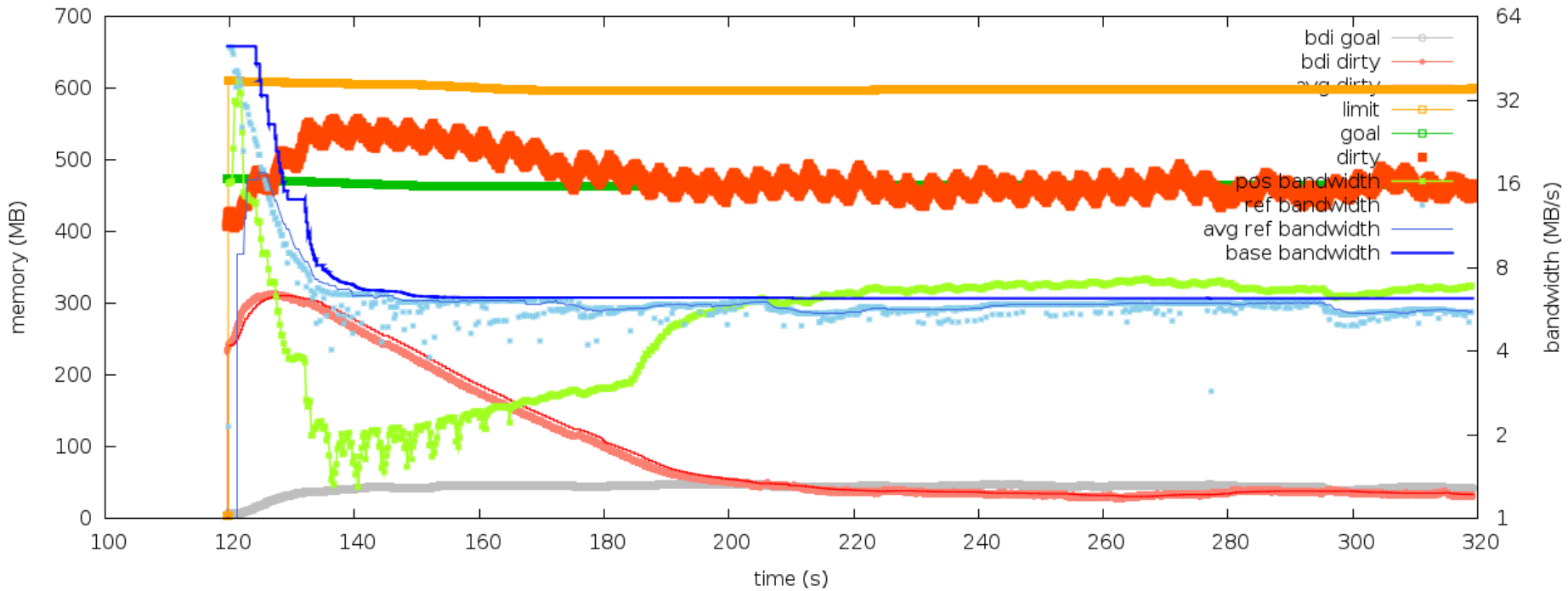


Over-dirtying example

UKEY accumulated much more initial `dirty pages` than its `bdi goal`.

$\Longrightarrow$

Bring `dirty pages` down by throttling the task at $1/4$ write bandwidth.



xfs, 1 dd to UKEY + 1 dd to HDD, 3G mem, 2.6.39-rc2-dt7+

Reasonable feedback in all mem/bw combinations

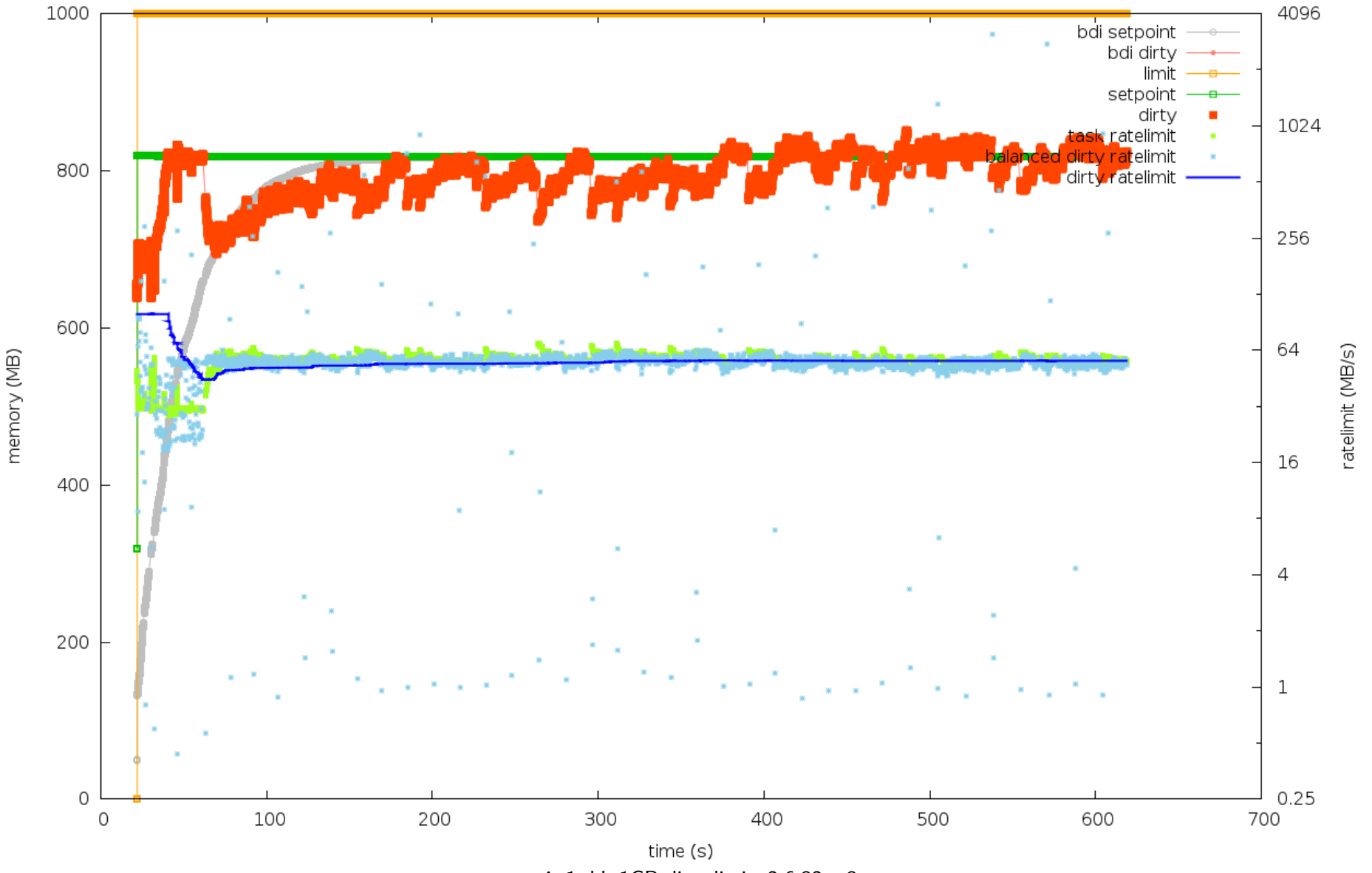
Policies are:

1. **global control line: adapts to dirtyable memory size**
2. **bdi control line: adapts to write bandwidth (except in JBOD)**

Designed for:

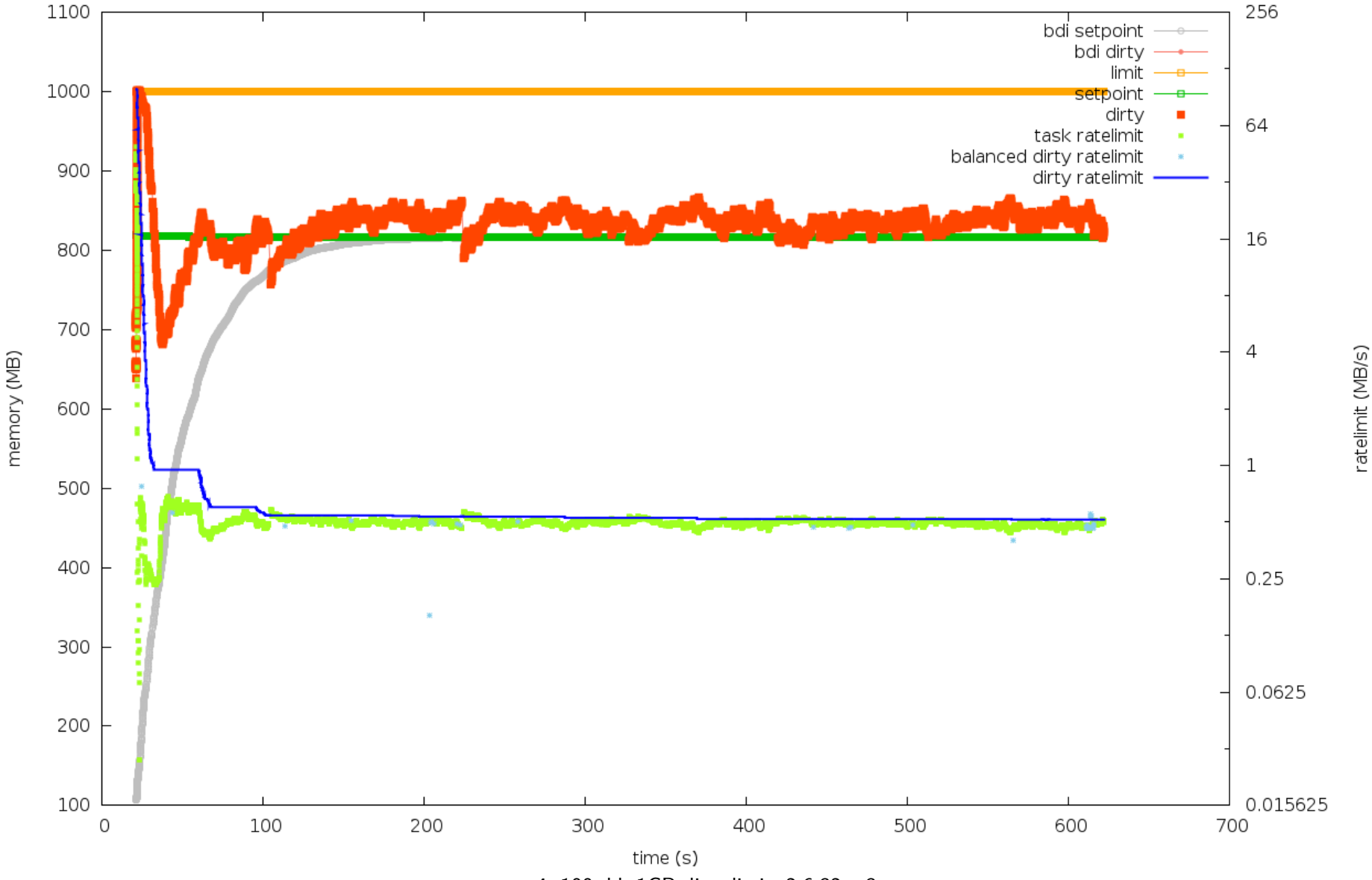
1. **small `dirty_setpoint`, large `write_bw`: global line will take control**
2. **large `dirty_setpoint`, small `write_bw`: bdi line will take control**
3. **JBOD case: `bdi_dirty_setpoint` fluctuates proportional to dirtyable memory size; do weak bdi position control**

1-disk case: fluctuations <= write bandwidth



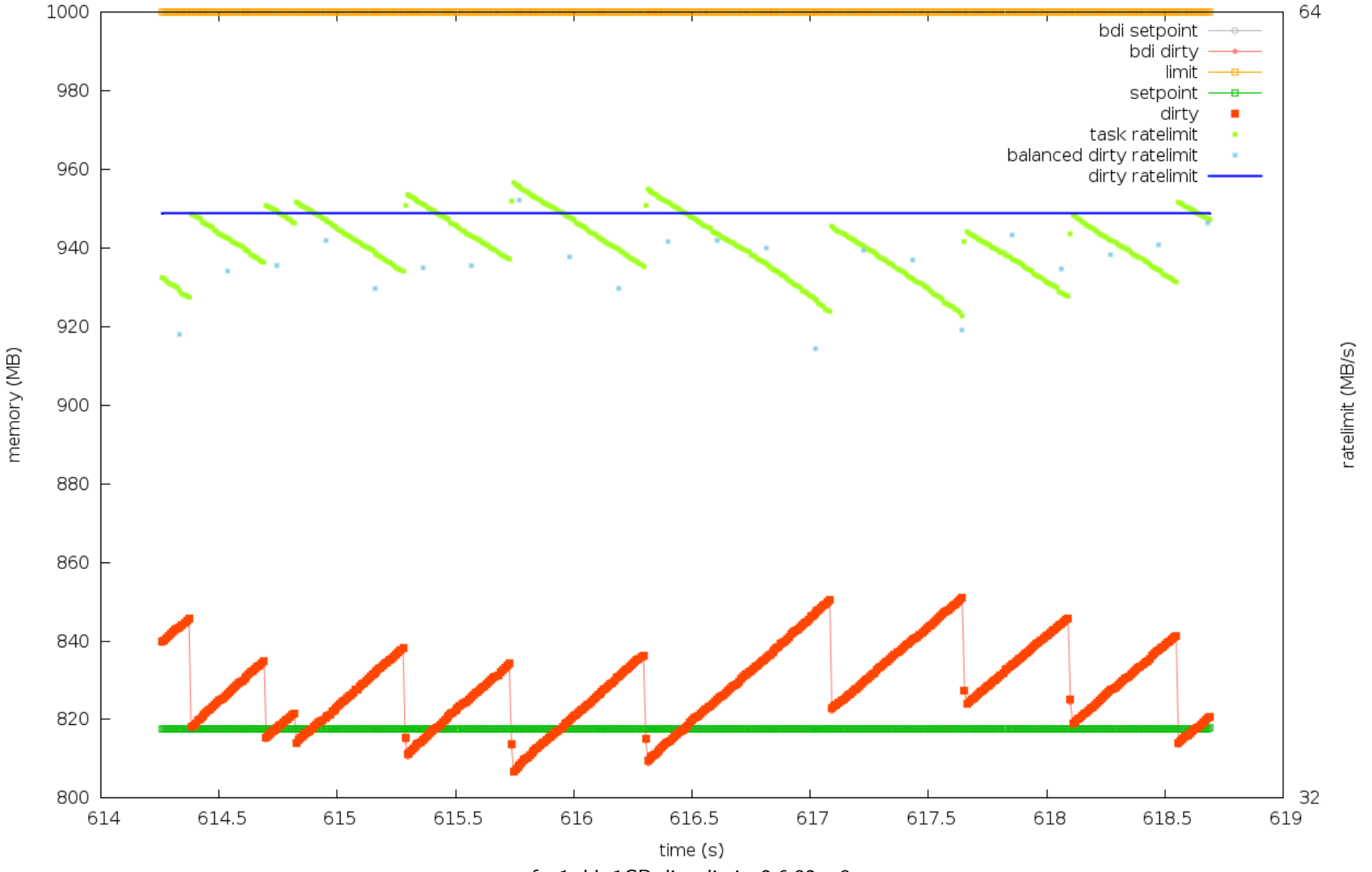
ext4, 1 dd, 1GB dirty limit, 2.6.32-rc3

1-disk case: fluctuations <= write bandwidth



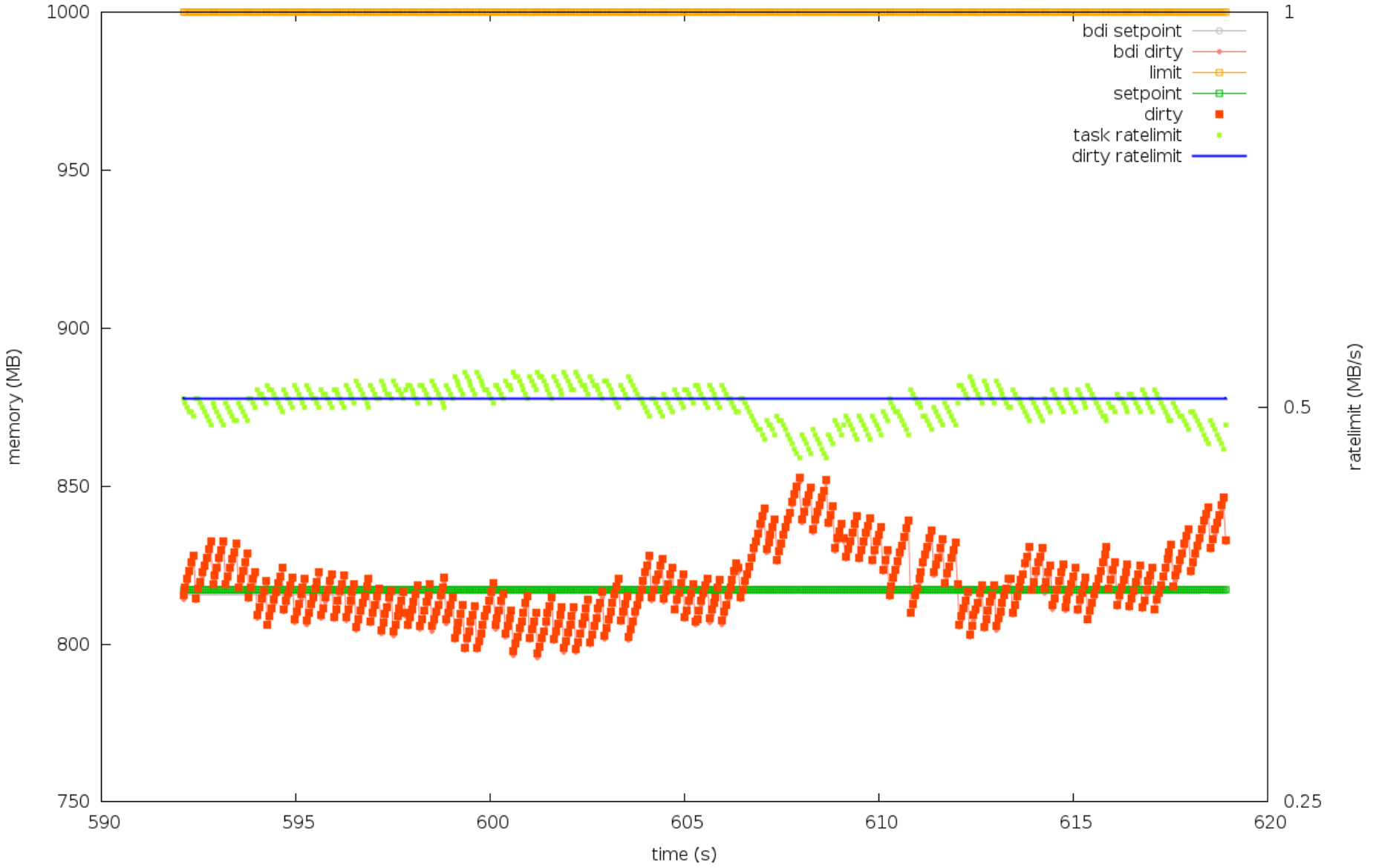
ext4, 100 dd, 1GB dirty limit, 2.6.32-rc3

1-disk case: fluctuations <= write bandwidth



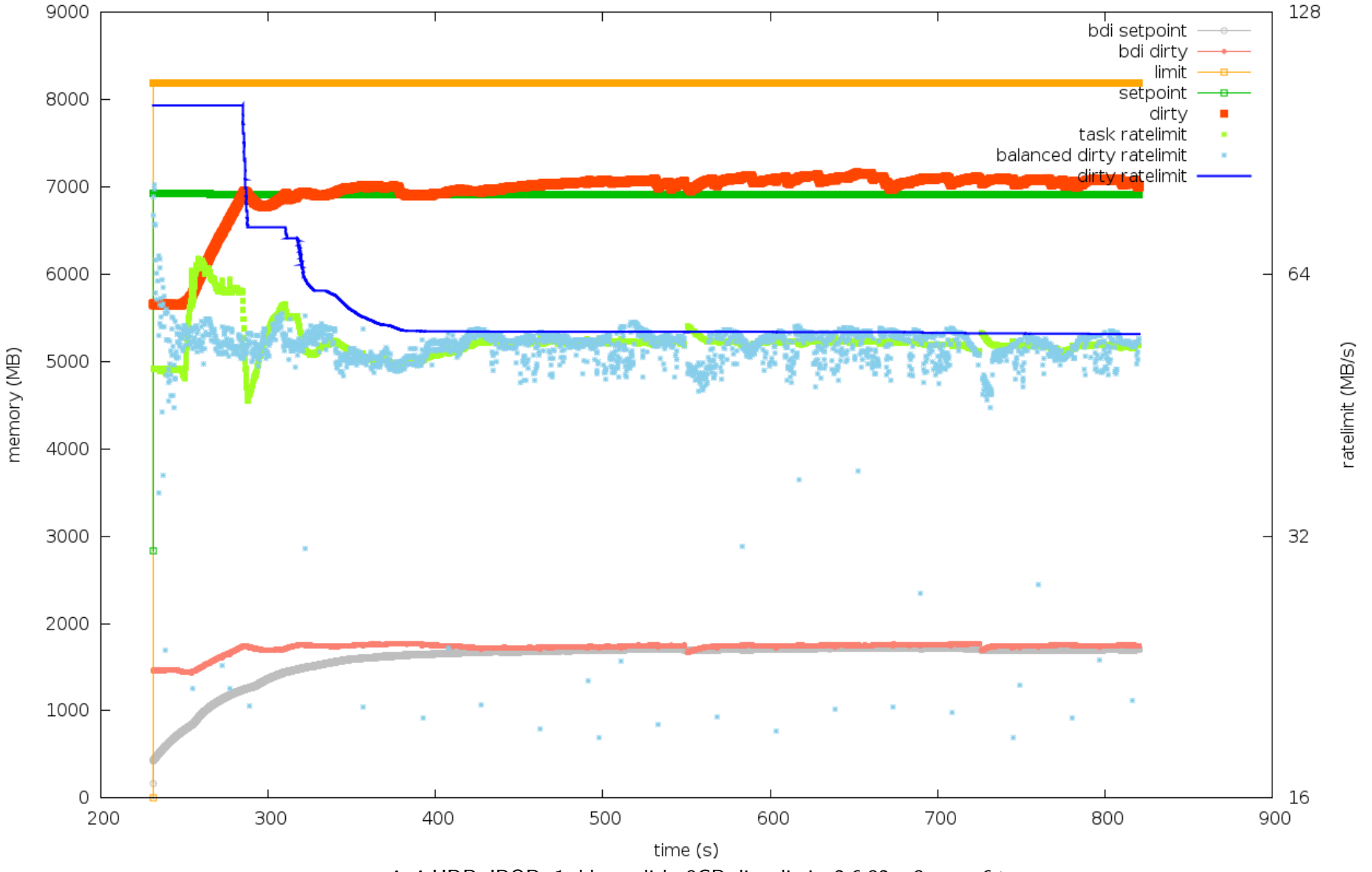
xfs, 1 dd, 1GB dirty limit, 2.6.32-rc3

1-disk case: fluctuations <= write bandwidth



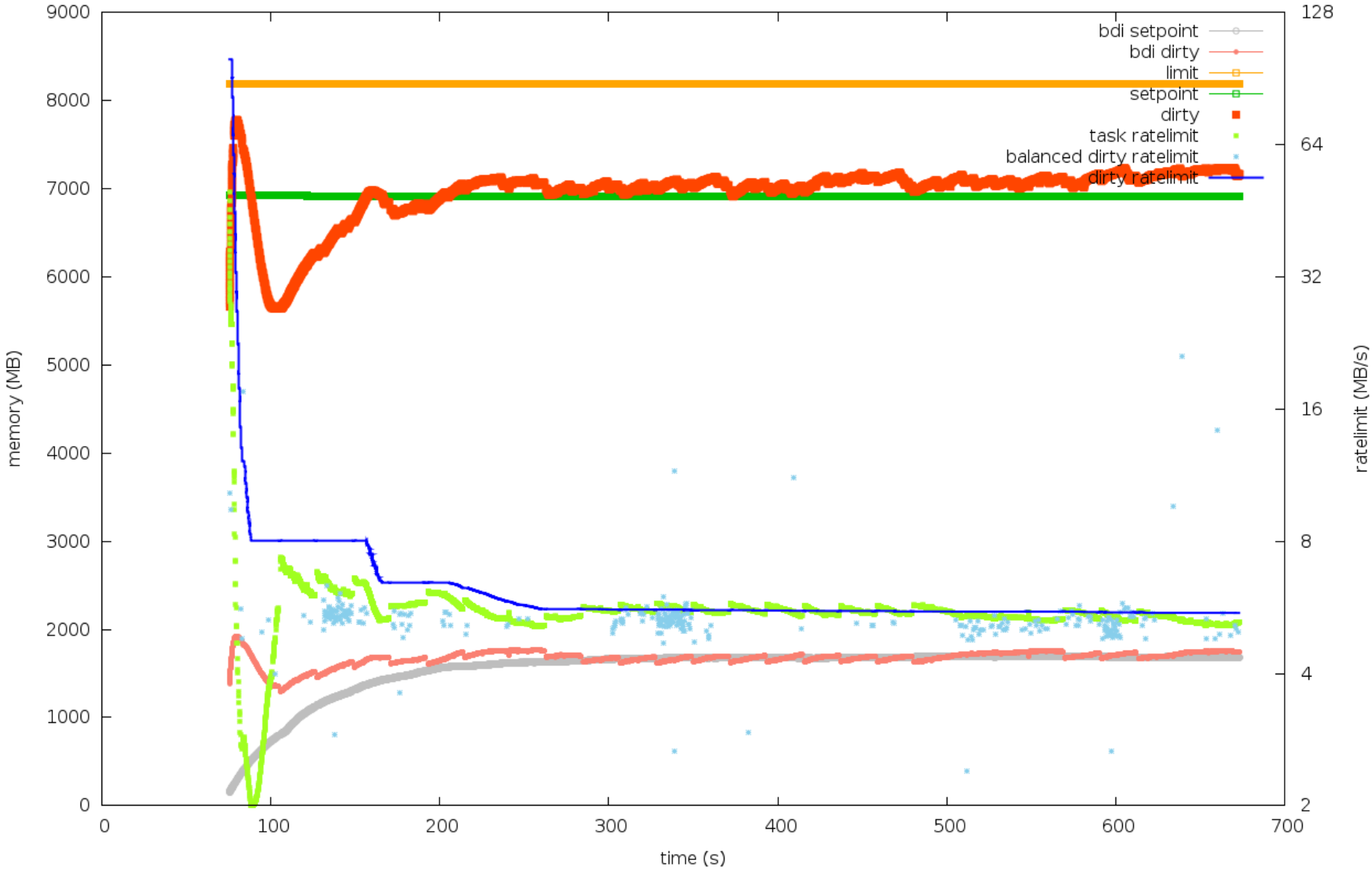
xfs, 100 dd, 1GB dirty limit, 2.6.32-rc3

JBOD case: large memory



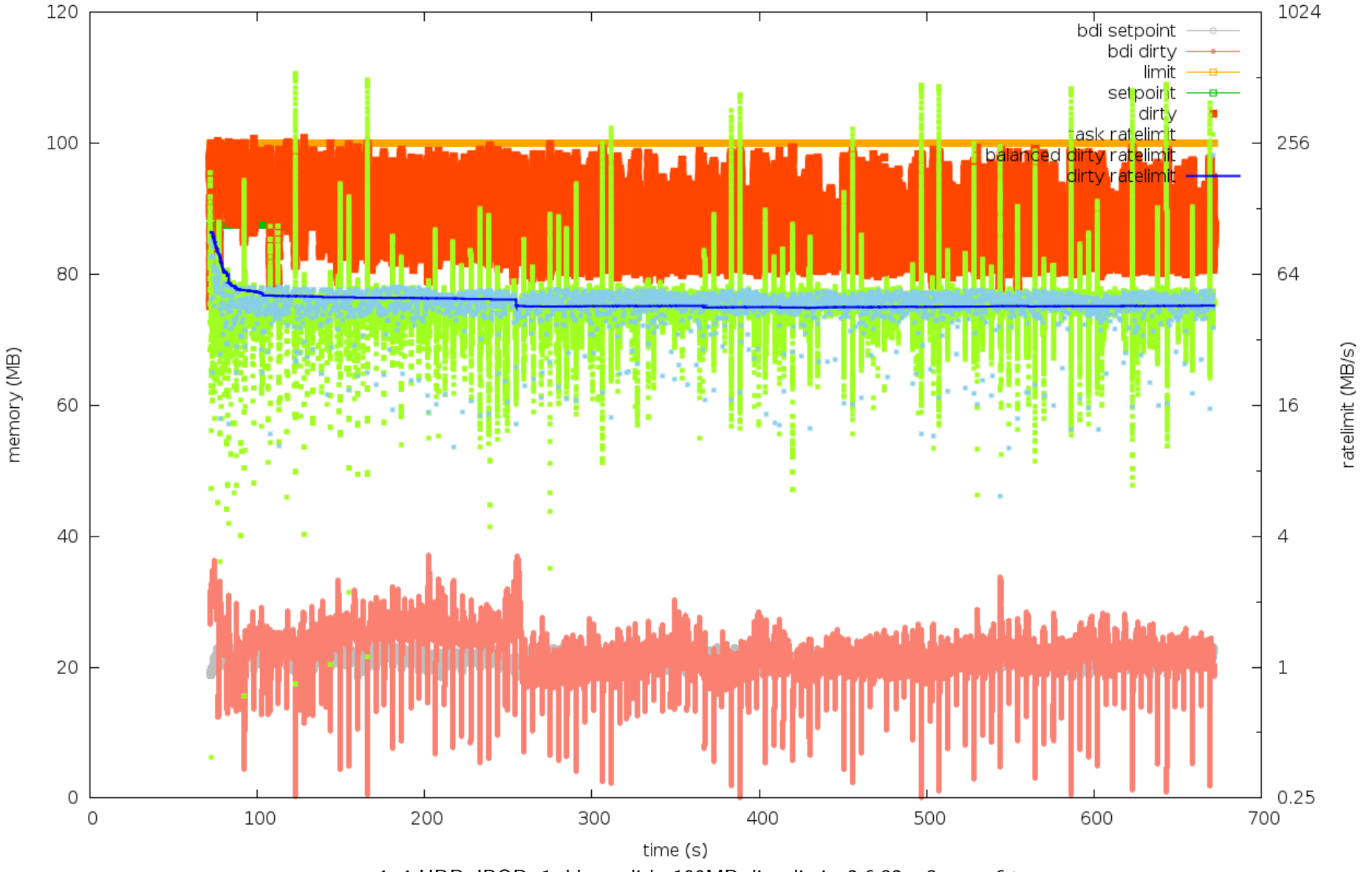
ext4, 4-HDD JBOD, 1 dd per disk, 8GB dirty limit, 2.6.32-rc3-pause6+

JBOD case: large memory



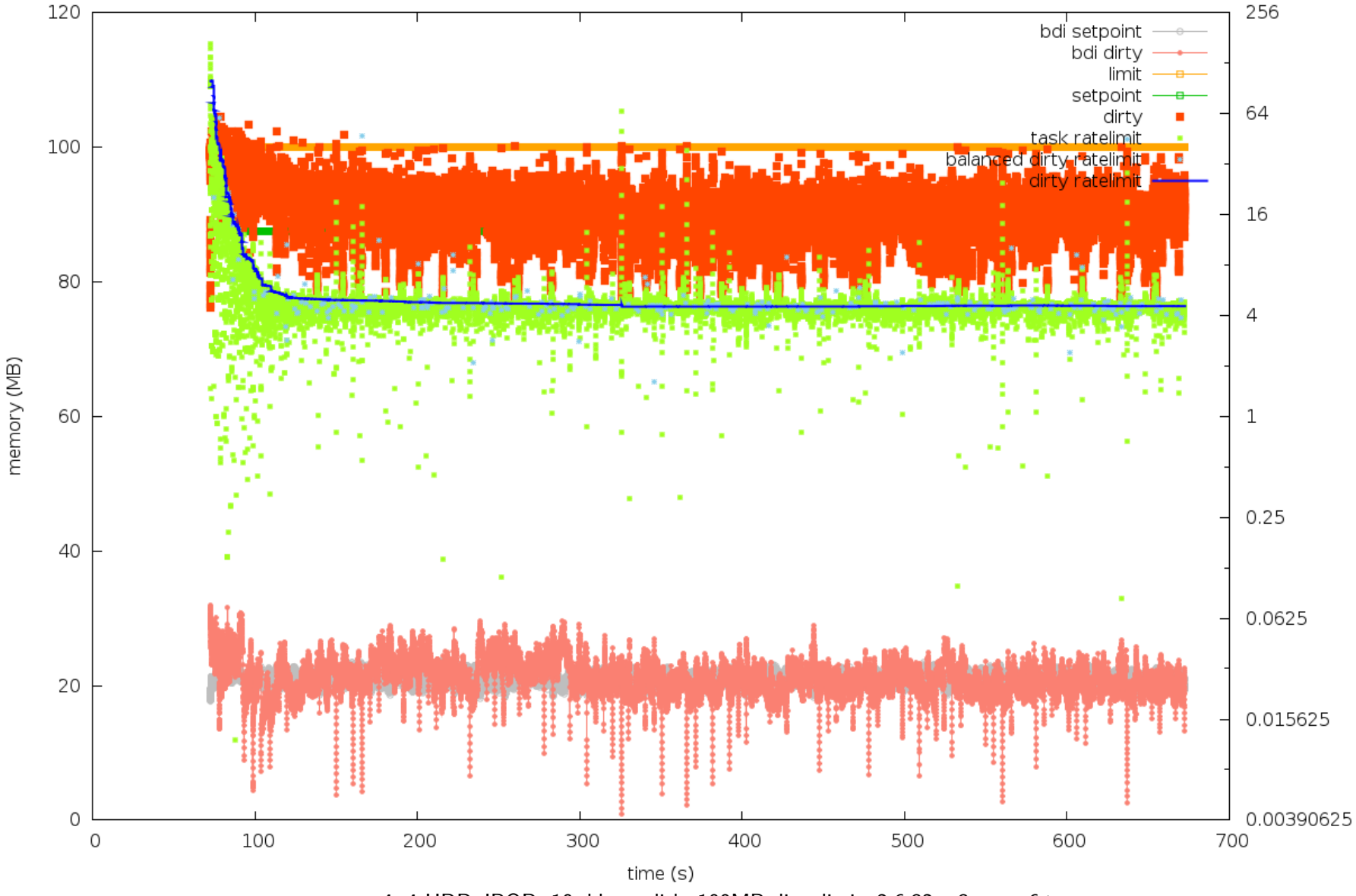
ext4, 4-HDD JBOD, 10 dd per disk, 8GB dirty limit, 2.6.32-rc3-pause6+

JBOD case: small memory



ext4, 4-HDD JBOD, 1 dd per disk, 100MB dirty limit, 2.6.32-rc3-pause6+

JBOD case: small memory



ext4, 4-HDD JBOD, 10 dd per disk, 100MB dirty limit, 2.6.32-rc3-pause6+

Roll it up

on write() syscall

```
balance_dirty_pages(pages_dirtied)
{
    task_ratelimit = bdi->dirty_ratelimit * bdi_position_ratio();
    sleep(pages_dirtied / task_ratelimit);
}
```

on every 200ms

```
bdi_update_dirty_ratelimit()
{
    N = dirty_rate / task_ratelimit;
    balanced_ratelimit = write_bw / N;

    update bdi->dirty_ratelimit torwards balanced_ratelimit,
                                regulated by task_ratelimit
}
```

What's next

1. **per-memcg dirty limits**

2. **`write_bps` throttling IO controller**

3. **proportional weight IO controller** (*challenging*)

Thank you!