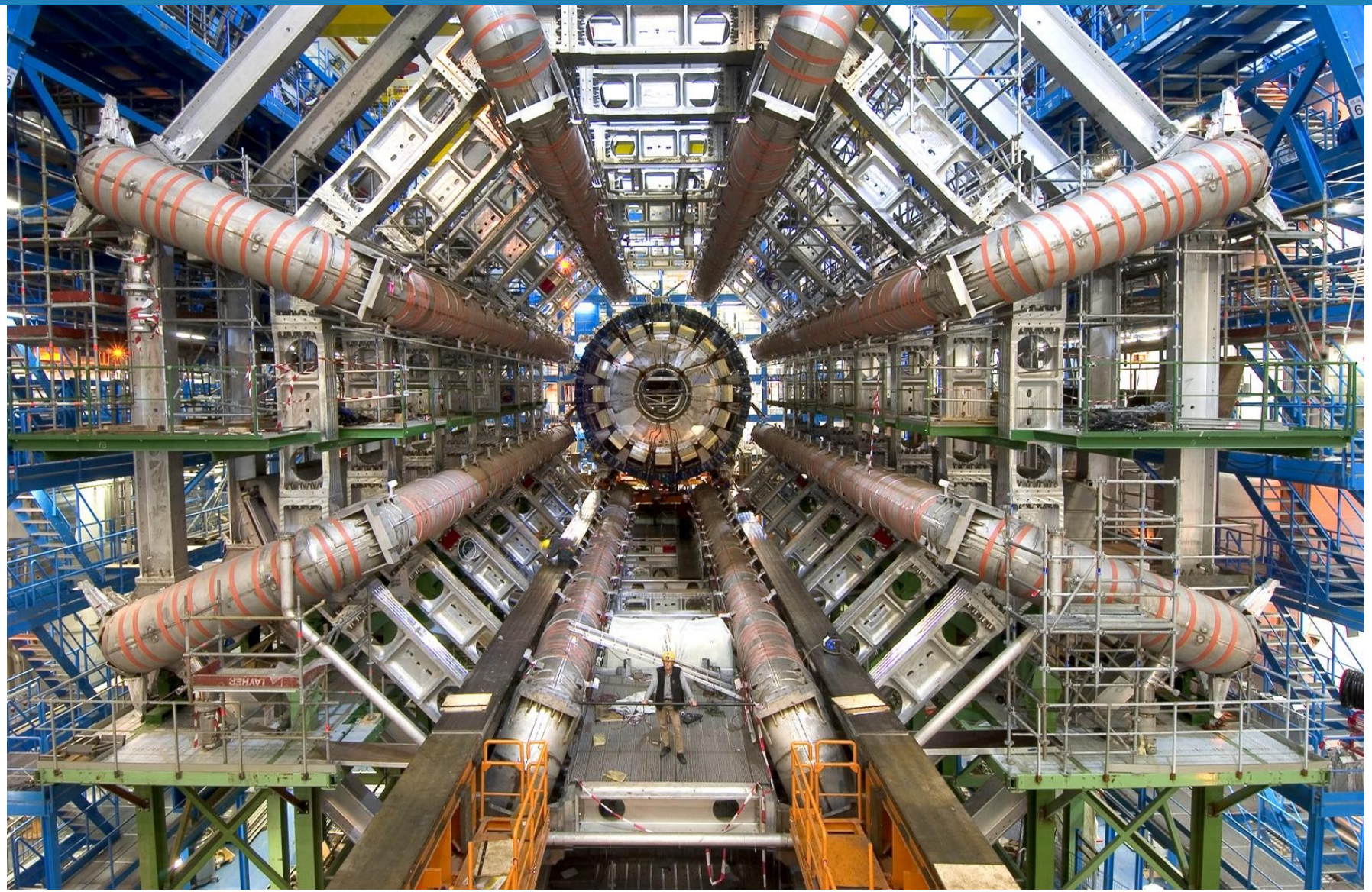
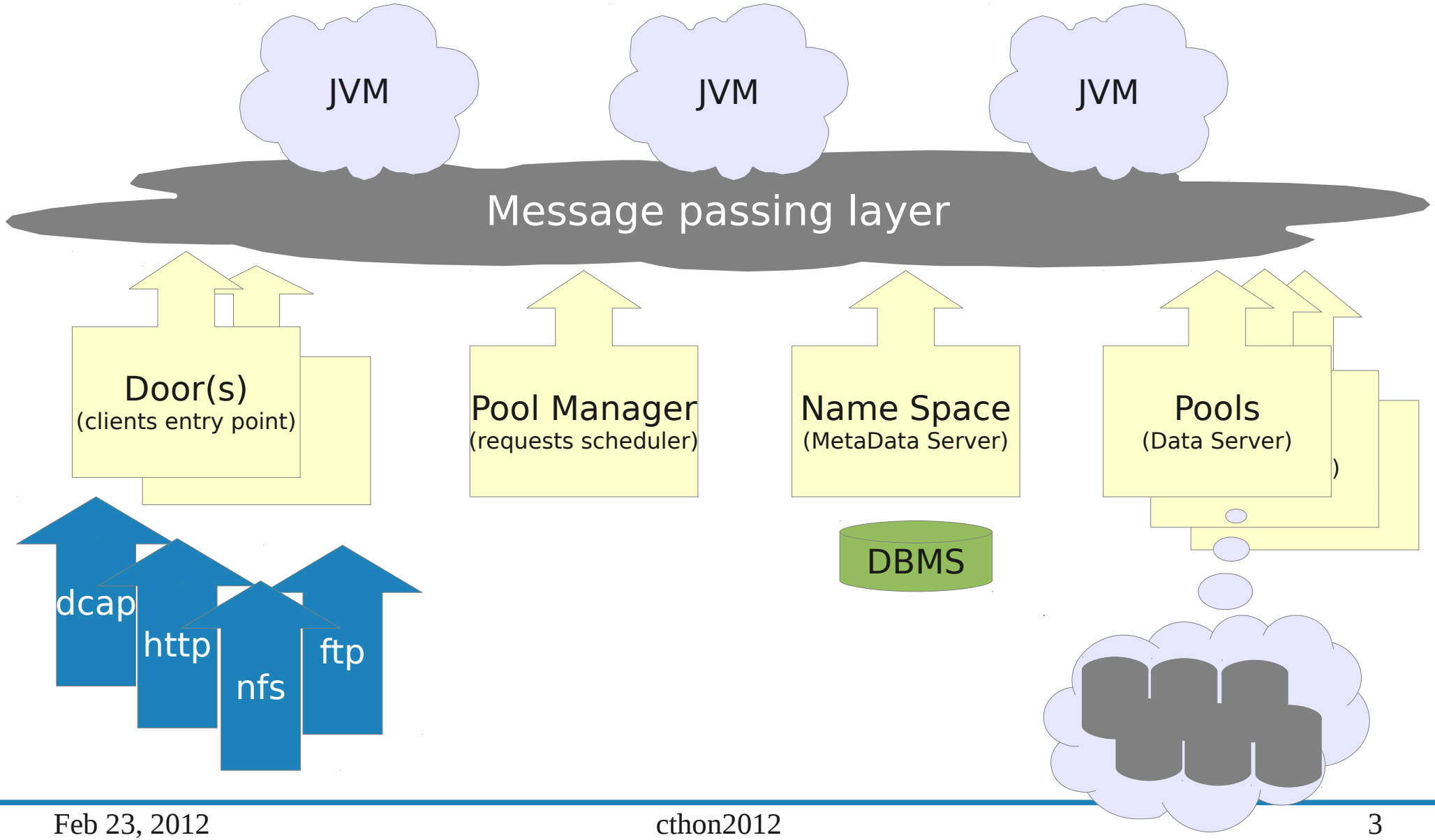
In 7500 lines to new RPC library

Tigran Mkrtchyan for dCache Team

Our toys



dCache in one slide



Why invent a new wheel?

- Not that many user space RPC libraries
- Not that many Java implementations
  - No bi-directional RPC support
  - No RPCSEC_GSS
  - Not up-to-date
- Official libtirpc not good enough
  - No bi-directional RPC
  - JAVA – C integration

Is it a square wheel?

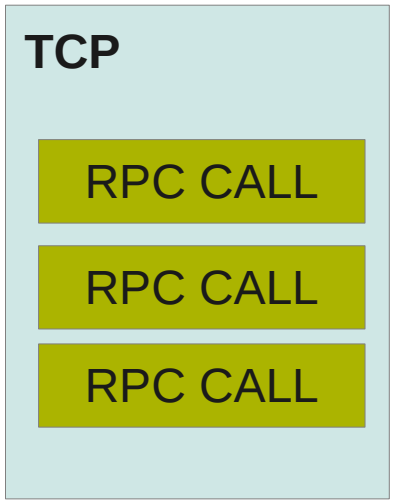
- High performance network IO is not an RPC/NFS requirements
  - Network components from GlassFish Application Server
- RFC 1831 and RFC 2203 compliant
- IPv6 support
- GSS handling comes from Java Run-time Environment
  - With comes with jre 6 AES128 and AES256
- Poll/epoll/select/p_threads handles by JVM
  - We use high level abstractions

Typical JAVA way

- Single thread per connection
  - Thousand threads per server
- Request processed almost in a single thread
  - No thread fencing (till first shared resource)
- Simple to implement
  - Blocking reads
  - Blocking writes
  - Idle threads costs nothing (ok, 48k stack space)

RPC vs. Others

- Many protocols are request-reply based
- No new requests as long as no reply
-

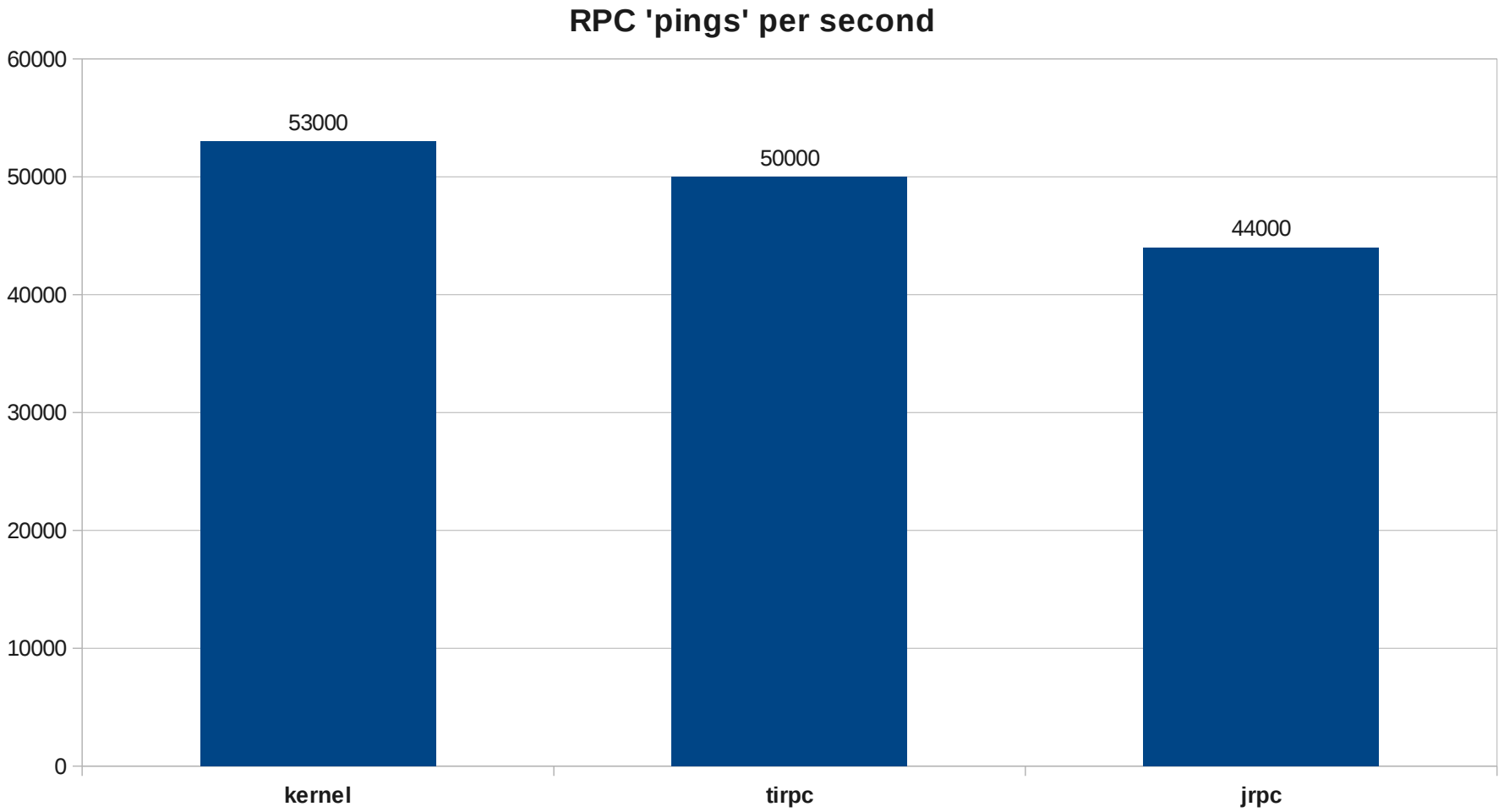


- Possible multiple independent requests
  - Even in one TCP package
- THE way to go for some workloads
  - High latency High bandwidth NFS access
    - UMICH <-> CERN

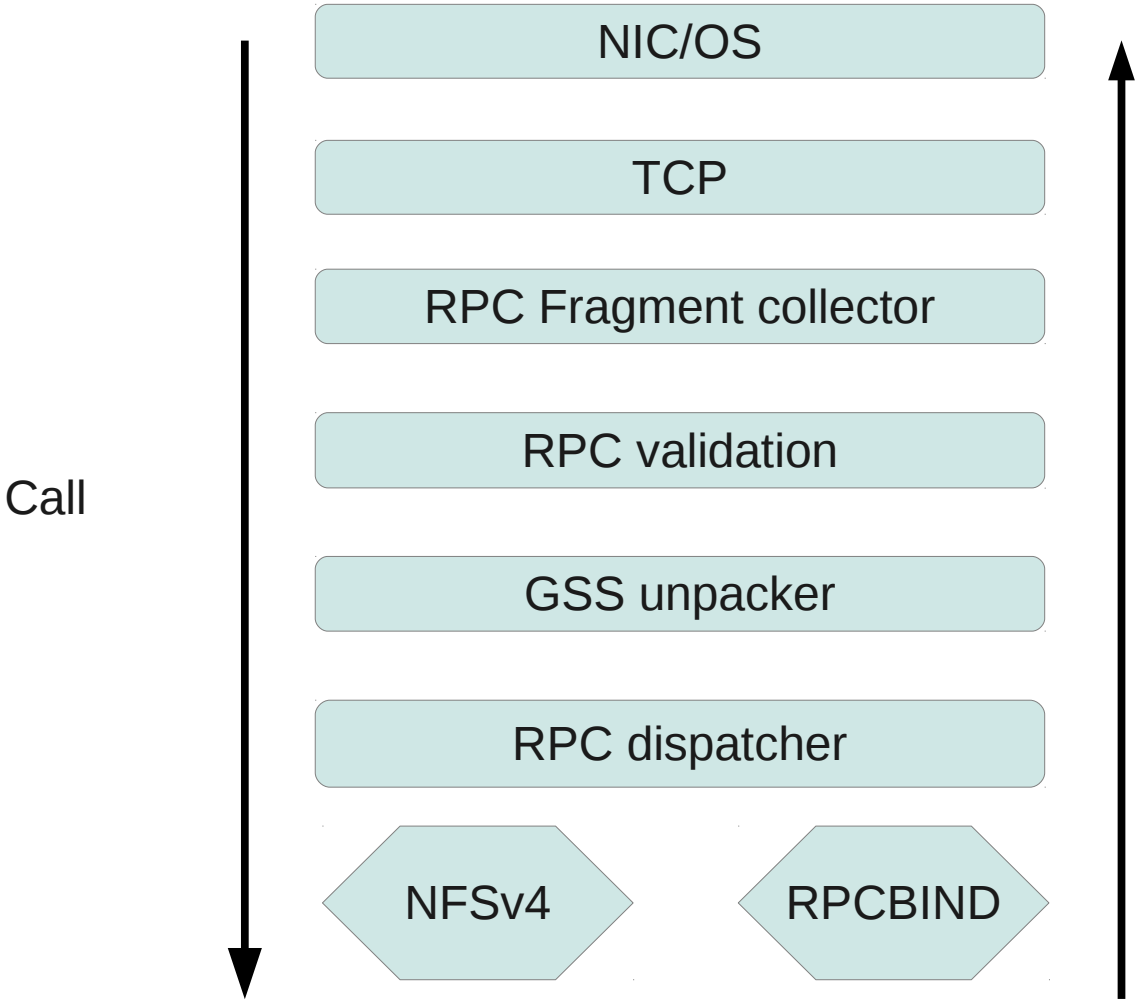
Our approach

- Poll of IO threads
  - Typically 2x#Cores
- Pool of worker threads
- Processing per PRC packet
  - No binding to network connection
- Event based
  - doOnRead if bytes arrived
  - doOnWrite if bytes sent

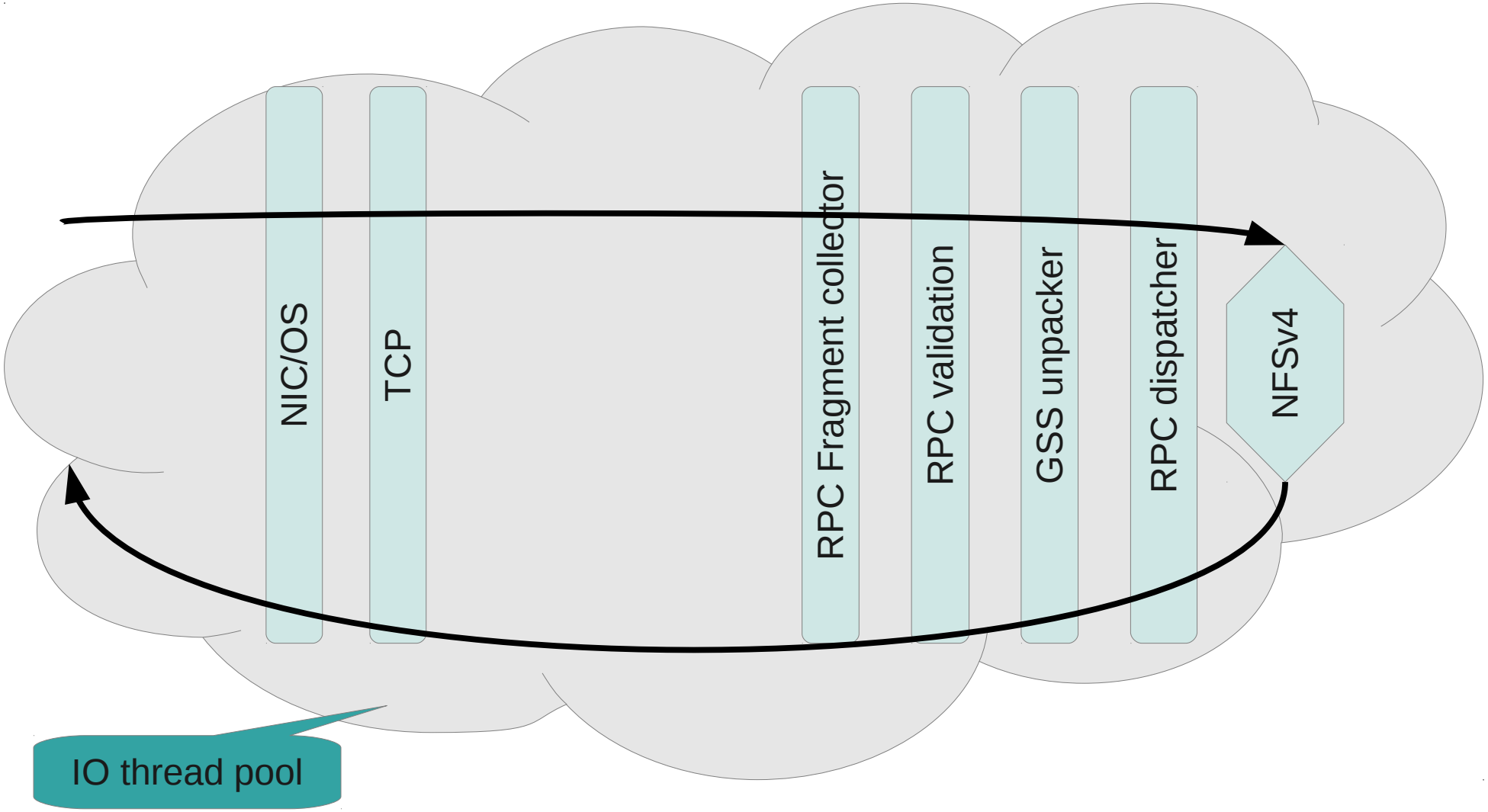
What about performance?



Chain of responsibilities

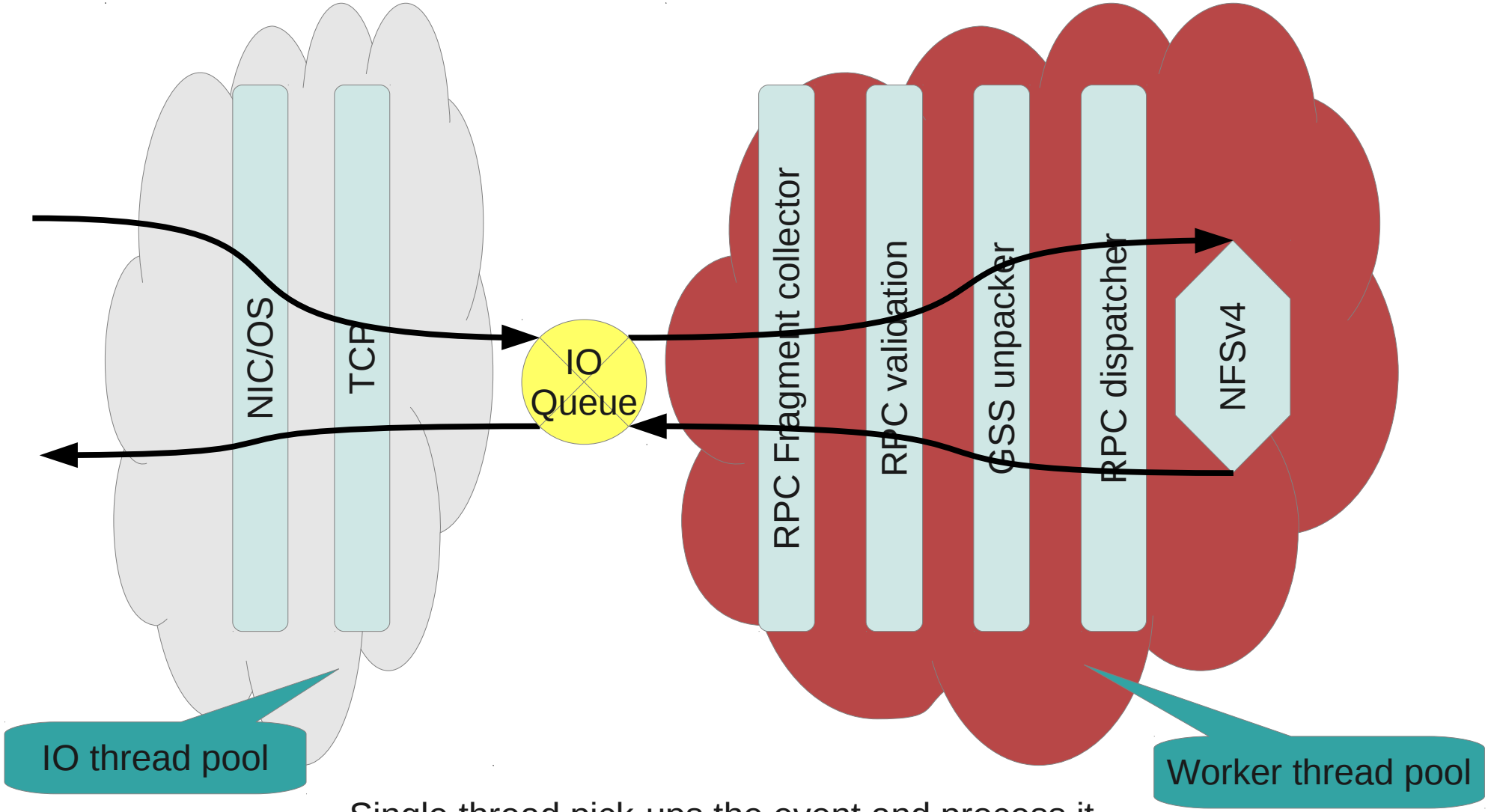


IO strategy: Same Thread



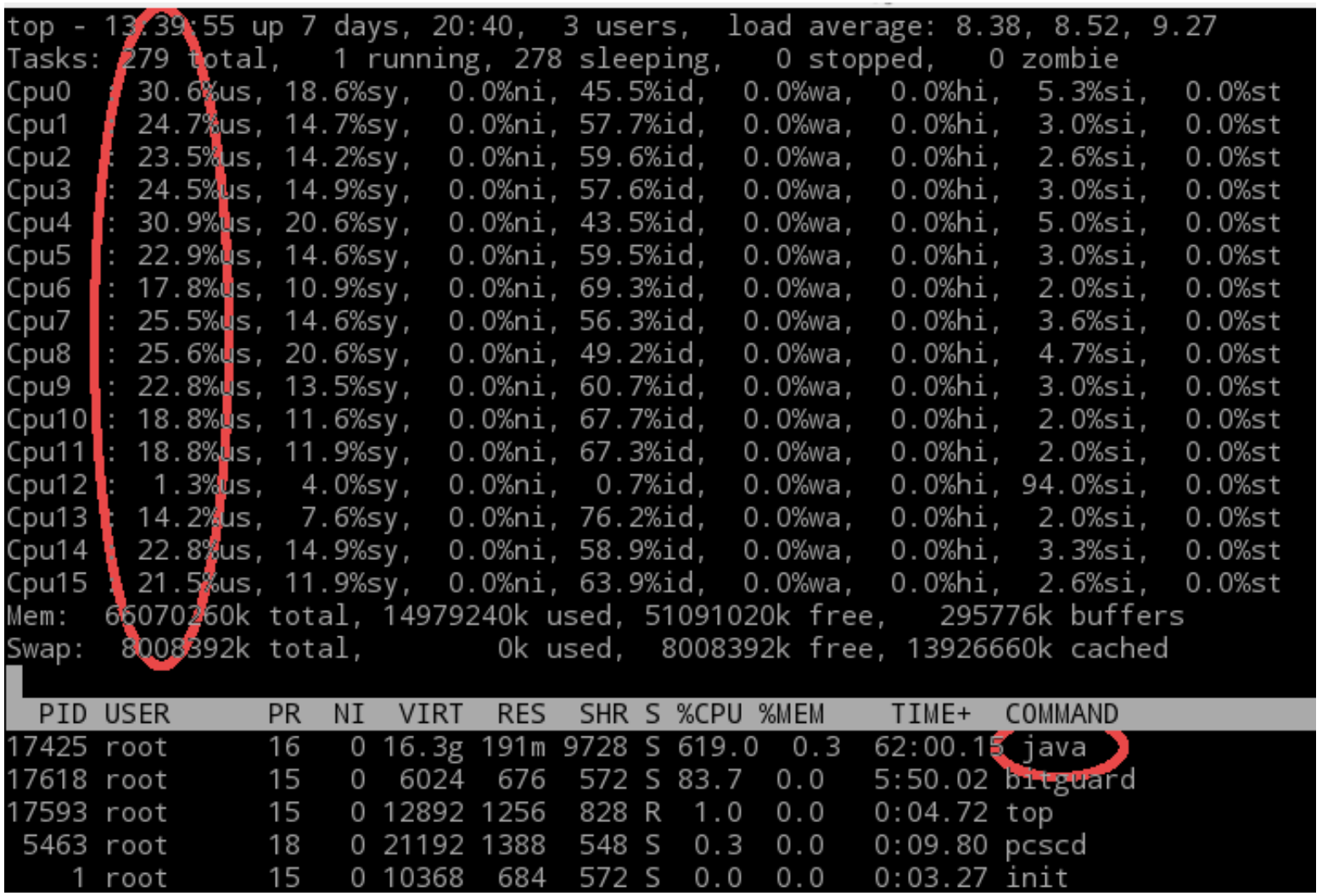
Single thread pick-ups the event and process it.

IO strategy: Worker Thread



Single thread pick-ups the event and process it.

Multi-Core

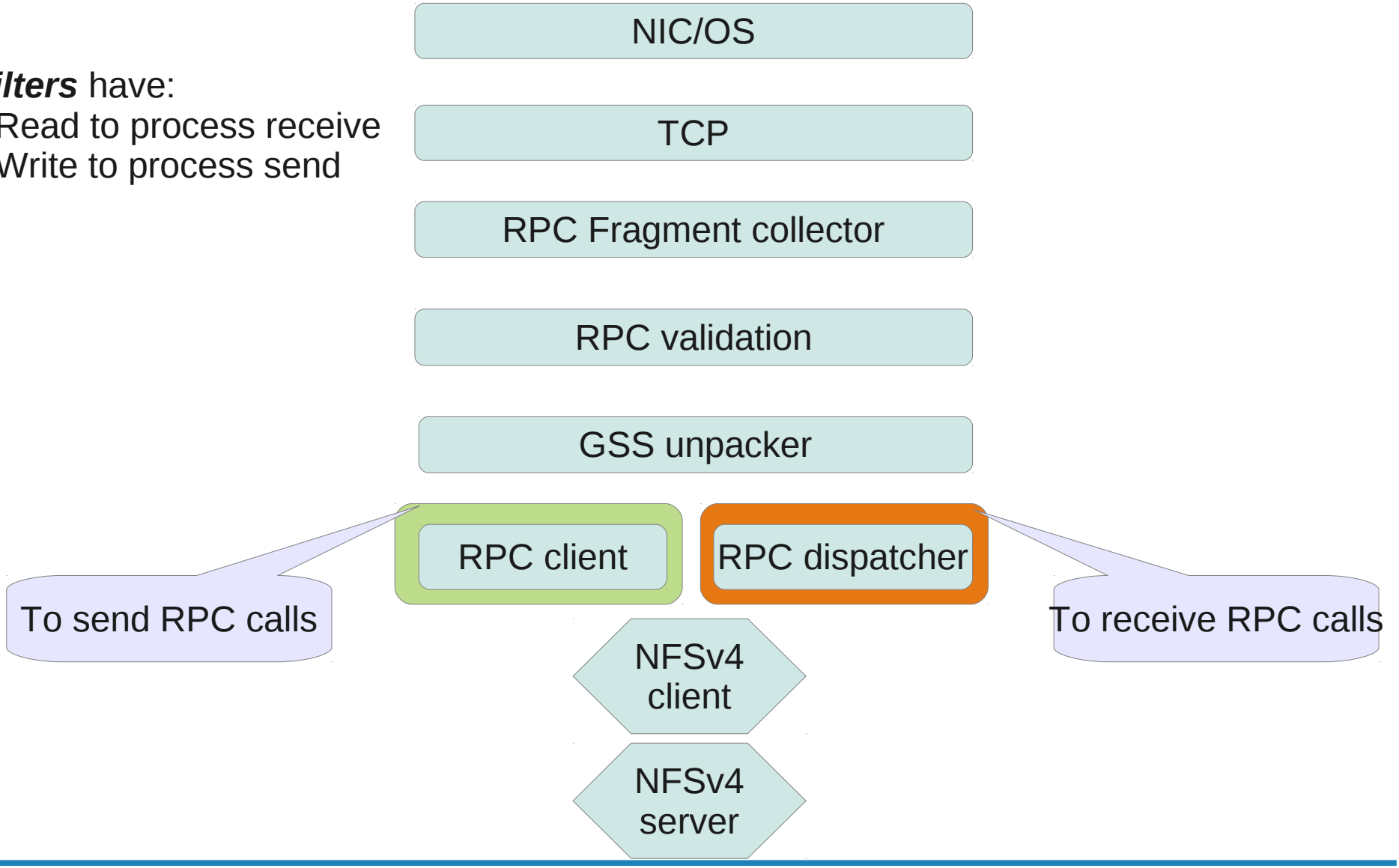

top - 13:39:55 up 7 days, 20:40, 3 users, load average: 8.38, 8.52, 9.27
Tasks: 279 total, 1 running, 278 sleeping, 0 stopped, 0 zombie
Cpu0 : 30.6%us, 18.6%sy, 0.0%ni, 45.5%id, 0.0%wa, 0.0%hi, 5.3%si, 0.0%st
Cpu1 : 24.7%us, 14.7%sy, 0.0%ni, 57.7%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu2 : 23.5%us, 14.2%sy, 0.0%ni, 59.6%id, 0.0%wa, 0.0%hi, 2.6%si, 0.0%st
Cpu3 : 24.5%us, 14.9%sy, 0.0%ni, 57.6%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu4 : 30.9%us, 20.6%sy, 0.0%ni, 43.5%id, 0.0%wa, 0.0%hi, 5.0%si, 0.0%st
Cpu5 : 22.9%us, 14.6%sy, 0.0%ni, 59.5%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu6 : 17.8%us, 10.9%sy, 0.0%ni, 69.3%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu7 : 25.5%us, 14.6%sy, 0.0%ni, 56.3%id, 0.0%wa, 0.0%hi, 3.6%si, 0.0%st
Cpu8 : 25.6%us, 20.6%sy, 0.0%ni, 49.2%id, 0.0%wa, 0.0%hi, 4.7%si, 0.0%st
Cpu9 : 22.8%us, 13.5%sy, 0.0%ni, 60.7%id, 0.0%wa, 0.0%hi, 3.0%si, 0.0%st
Cpu10 : 18.8%us, 11.6%sy, 0.0%ni, 67.7%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu11 : 18.8%us, 11.9%sy, 0.0%ni, 67.3%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu12 : 1.3%us, 4.0%sy, 0.0%ni, 0.7%id, 0.0%wa, 0.0%hi, 94.0%si, 0.0%st
Cpu13 : 14.2%us, 7.6%sy, 0.0%ni, 76.2%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu14 : 22.8%us, 14.9%sy, 0.0%ni, 58.9%id, 0.0%wa, 0.0%hi, 3.3%si, 0.0%st
Cpu15 : 21.5%us, 11.9%sy, 0.0%ni, 63.9%id, 0.0%wa, 0.0%hi, 2.6%si, 0.0%st
Mem: 66070260k total, 14979240k used, 51091020k free, 295776k buffers
Swap: 8008392k total, 0k used, 8008392k free, 13926660k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
17425 root 16 0 16.3g 191m 9728 S 619.0 0.3 62:00.15 java
17618 root 15 0 6024 676 572 S 83.7 0.0 5:50.02 bitguard
17593 root 15 0 12892 1256 828 R 1.0 0.0 0:04.72 top
5463 root 18 0 21192 1388 548 S 0.3 0.0 0:09.80 pcscd
1 root 15 0 10368 684 572 S 0.0 0.0 0:03.27 init

All together

```
RpcDispatchable nfs4 = new NFSServerV41(....);
OncRpcSvc svc = new OncRpcSvcBuilder()
        .withTCP()
        .withAutoPublish()
        .withPort(2049)
        .withSameThreadIoStrategy()
        .build();
svc.register(nfs4_prot.NFS4_PROGRAM, nfs4);
svc.start();
```

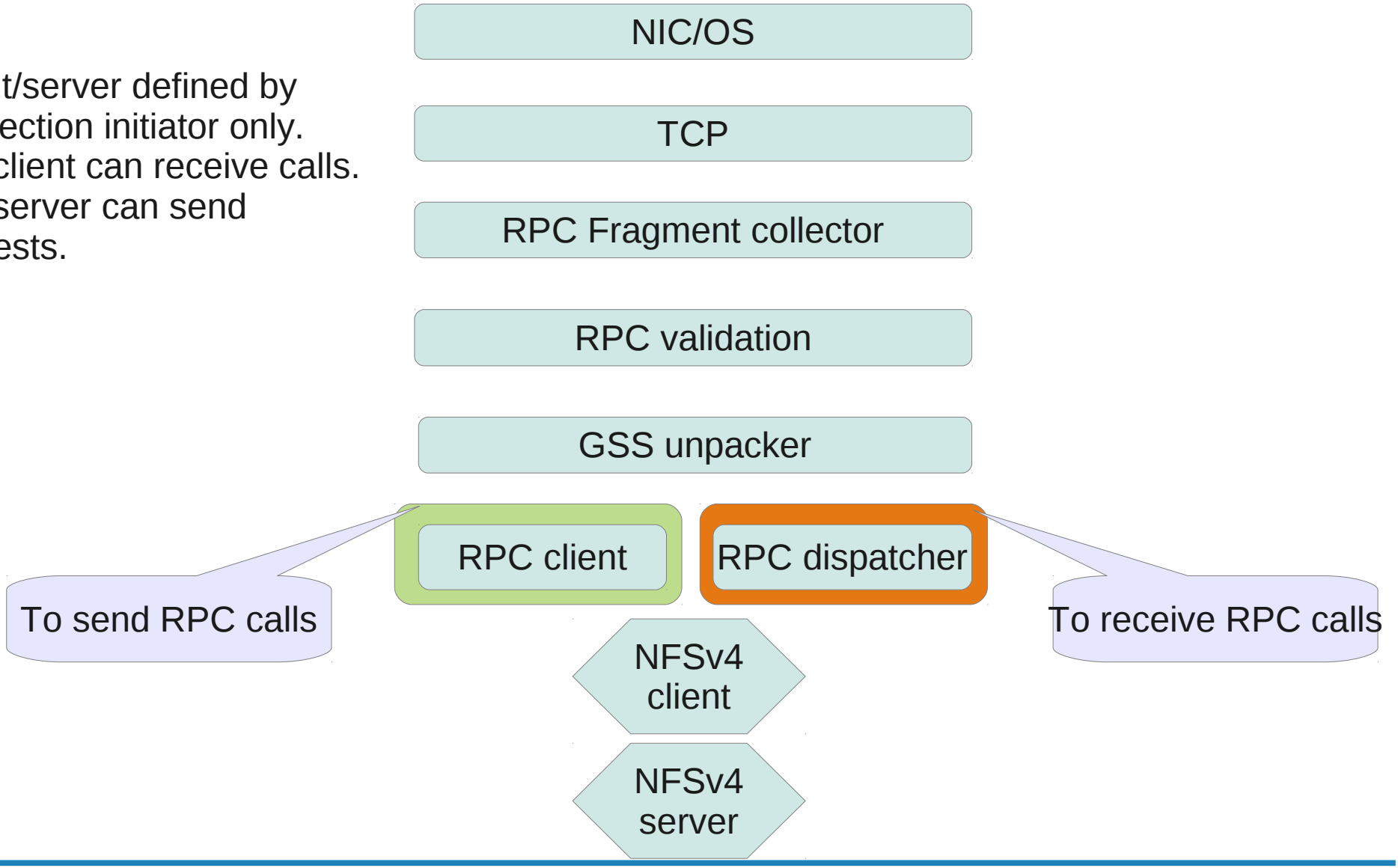
Code re-use (and much more)

- All ***Filters*** have:
  - onRead to process receive
  - onWrite to process send



Bi-directional RPC

- Client/server defined by connection initiator only.
- Any client can receive calls.
- Any server can send requests.



Ready to use by others

- Split ed into independent library
- Hosted on http://code.google.com/p/nio-jrpc/
- Licensed with LGPLv2