CMW – LHC-era controls middleware

CMW – **C**ontrols **M**iddle**w**are from BE-CO

DAQ@LHC workshop, 13th April 2016

Wojciech Sliwinski
for the BE-CO Middleware team:
Joel Lauener, Felix Medina Ch.
Wojciech Zadlo, Vitaliy Rapp (GSI)

Agenda

- What is Middleware?
- Evolution of middleware technologies
- CMW – Controls Middleware from BE-CO
- Conclusions

About the speaker

- Full-time Software Engineer

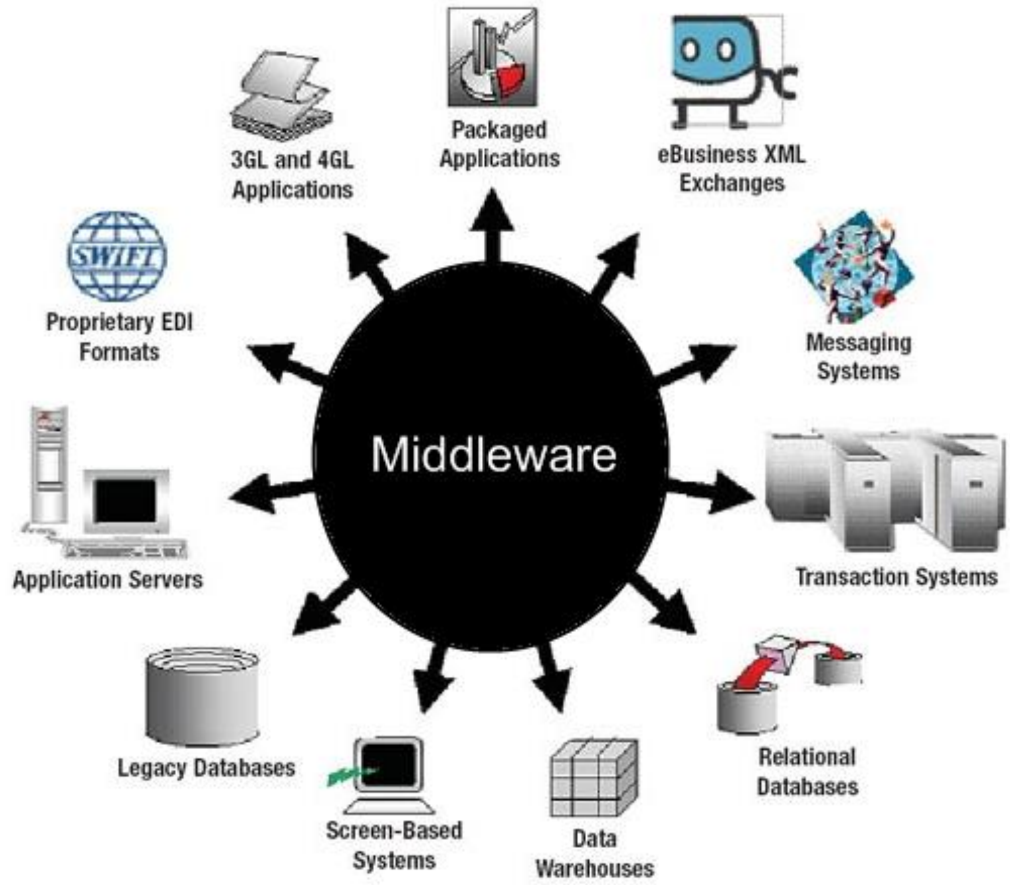
- Member of Beams dep., Controls group (BE-CO)
  - SRC (Software for Real-time and Communication) section
  - Leading Middleware team (5 people)
    - 2 Staff, 2 Fellows, 1 External (GSI)

- Before:
    - Fellow in IT-CS (LANDB database & tools)
    - Software Developer in ABB Research Center (Poland)
    - Tech. Student in FAP-AIS (EDH project)
    - Trainee in Rockwell Automation (Switzerland)

email: Wojciech.Sliwinski@cern.ch / w.s@cern.ch

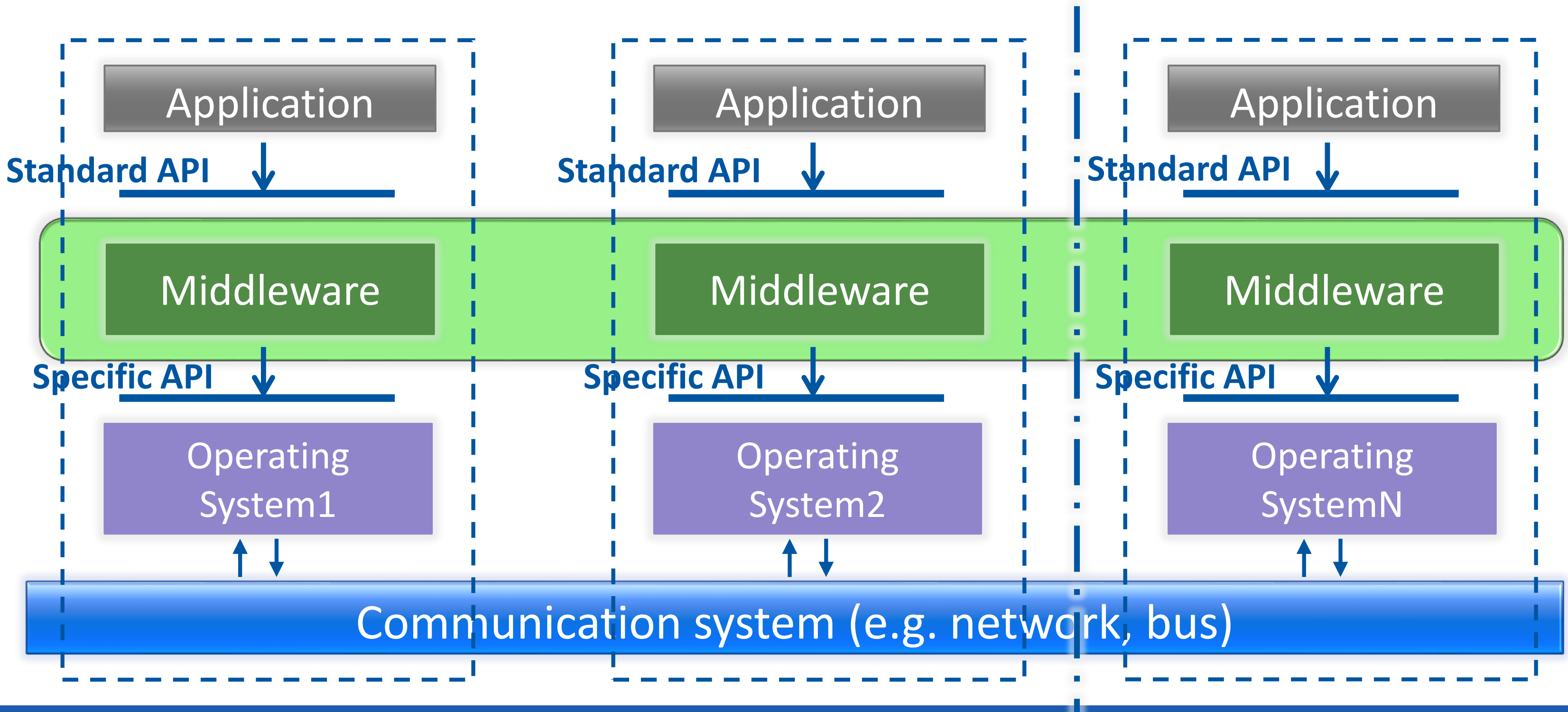
Why Middleware?

- How do we build/connect applications over a network?
- How do we facilitate Distributed Computing (2-tier, …, n-tier)?
- How to support the heterogeneous environments?



What is Middleware?

- *Software, which allows an application to interoperate with other software, without requiring the user to understand and to code the low-level operations required to achieve interoperability*
  - Software layer between OS and the applications
  - Hides complexity & heterogenity of distributed system
  - Handles issues related to OS, network protocols & hardware platforms



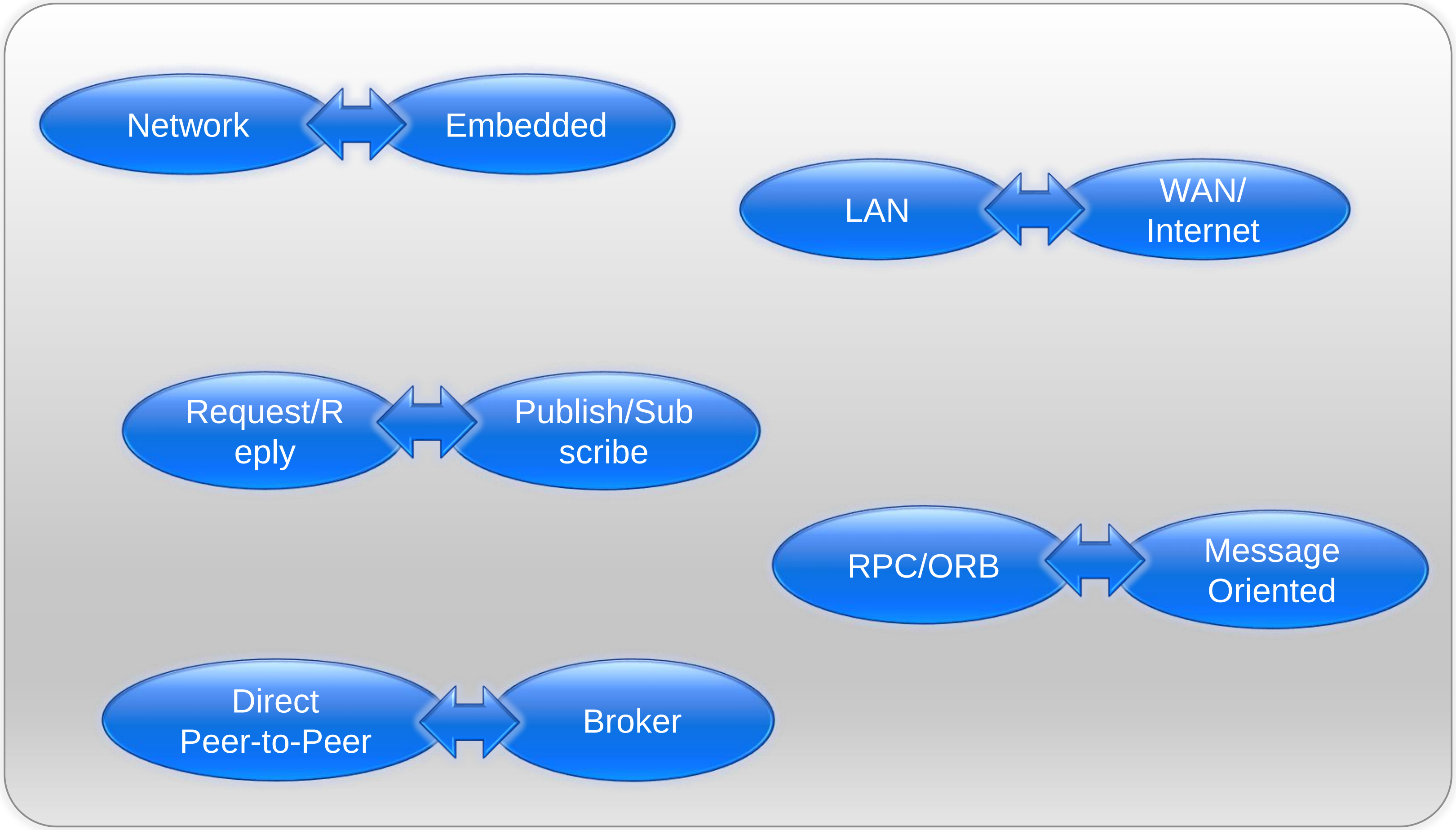
Aspects of Middleware

- Middleware provides support for:
  - Naming, Location, Service discovery, Replication
  - Protocol handling, Communication faults, QoS
  - Synchronisation, Concurrency, Failover, Scalability
  - Access control, Authentication

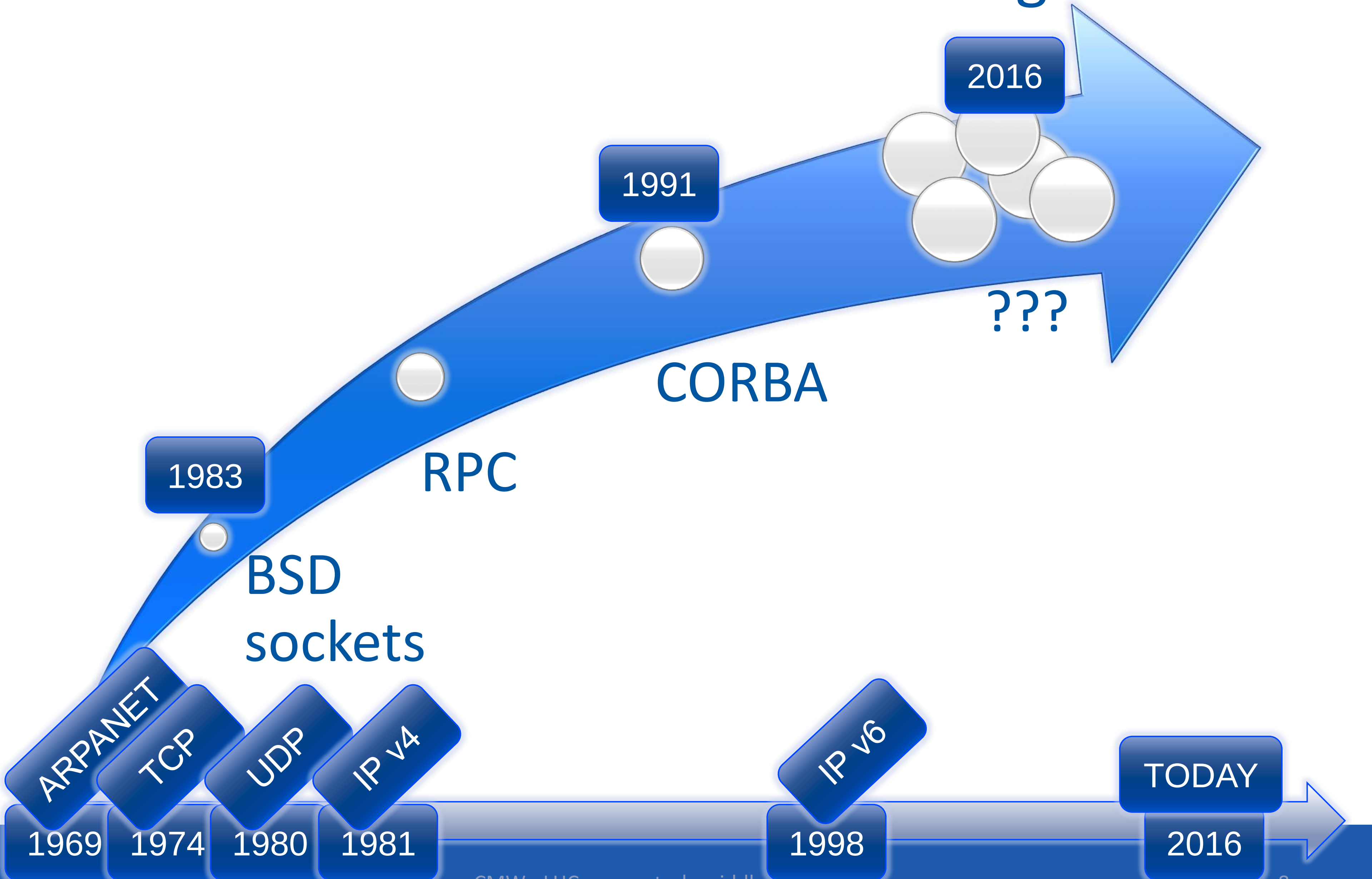
- Middleware dimensions:

• Request-Reply	vs.	Asynchronous Messaging
• Language-specific	vs.	Language-independent
• Proprietary	vs.	Standards-based
• Small-scale	vs.	Large-scale
• Tightly-coupled	vs.	Loosely-coupled components

Middleware – large domain … which to choose?



Evolution of middleware technologies



Dynamically evolving domain …

Peer-to-Peer

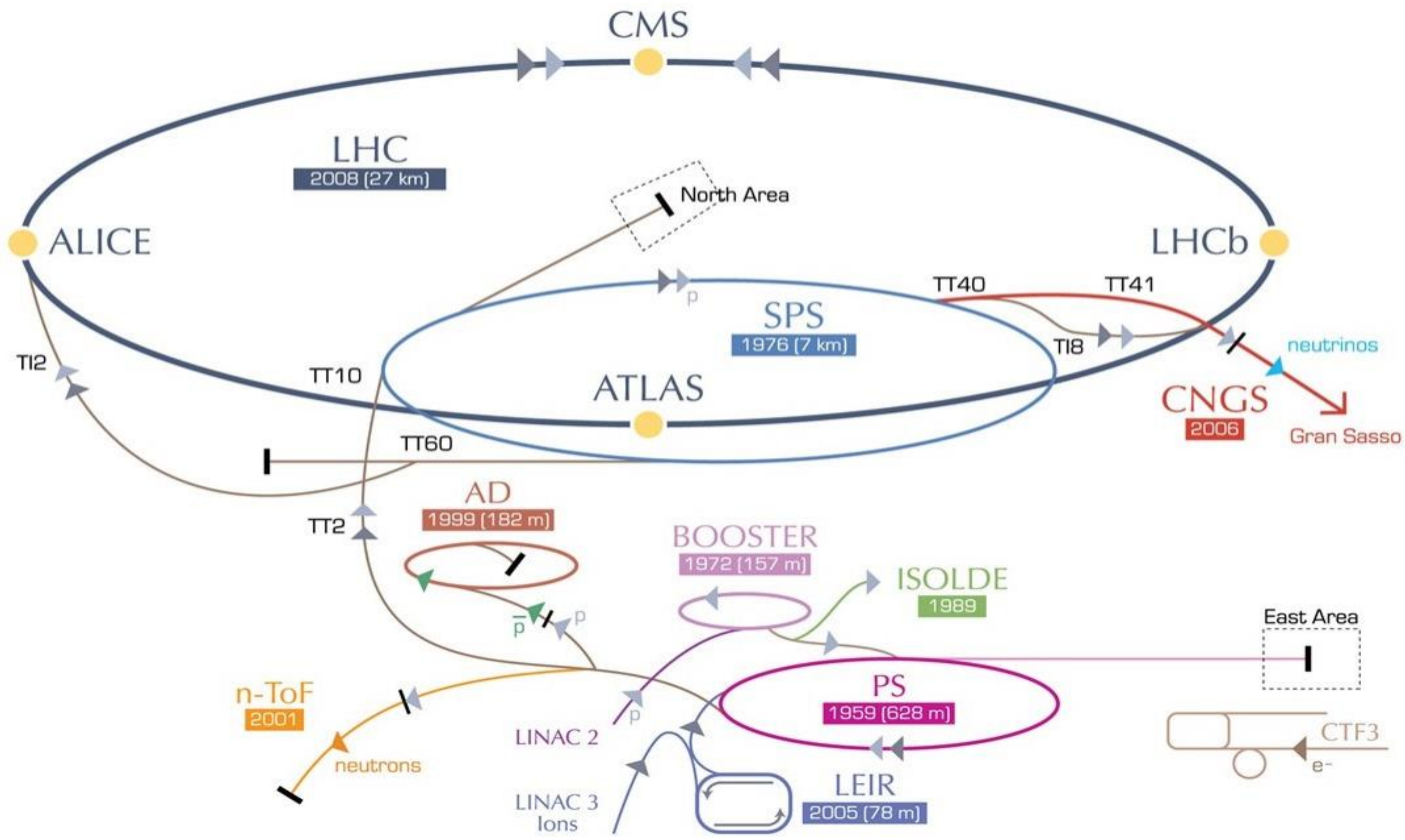
- CoreDX
- RTI DDS
- OpenSpliceDDS
- nanomsg
- ZeroMQ
- Thrift
- ZeroC Ice
- JacORB
- omniORB

Brokers

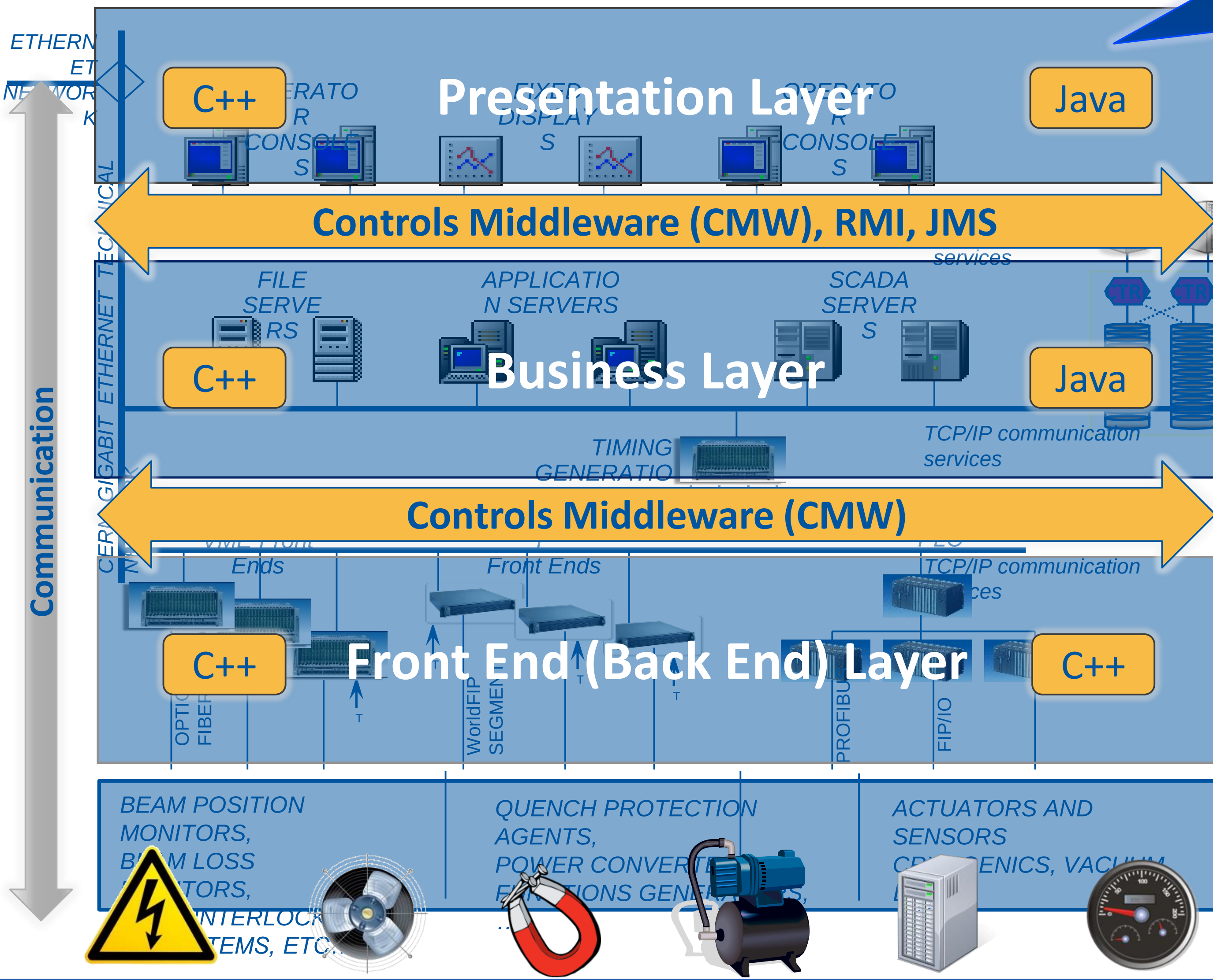
- Redis
- OpenAMQ
- ActiveMQ
- Apollo
- Kafka
- RabbitMQ
- HornetQ
- JoramMQ
- ThingMQ
- MQtt RSMB
- HiveMQ
- Mosquito
- Mosca
- Solace

CMW – Controls Middleware from BE-CO

Our clients: CERN Accelerators



Controls Software Architecture

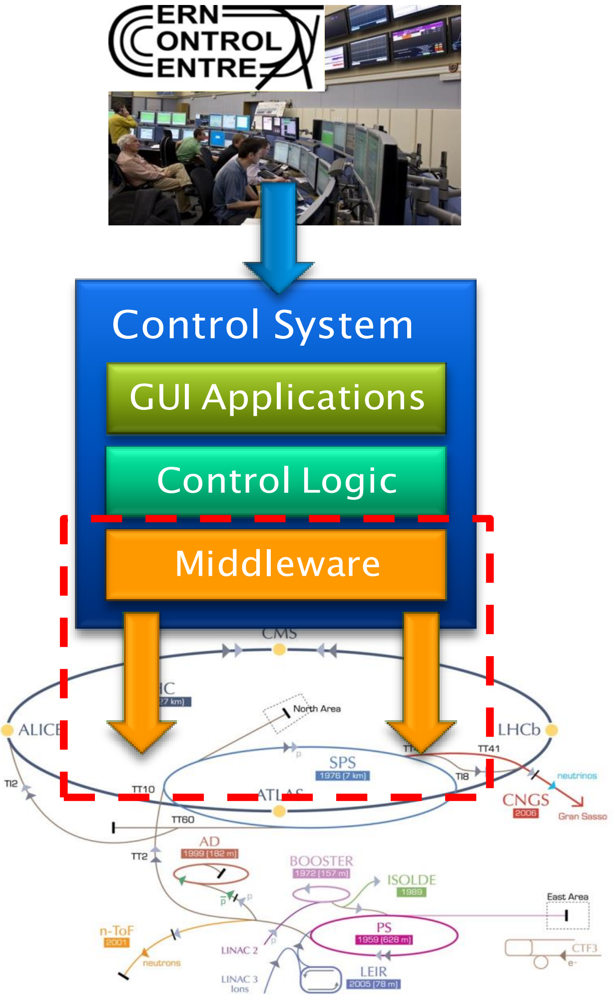


CMW Purpose:
Reliable and **scalable** transport of data between controls processes (Java & C++)

- ✓ 4'000 CMW servers
- ✓ 85'000 devices
- ✓ 2'000'000 IO-points
- ✓ Uses ZeroMQ & CORBA

CMW mandate & scope

- Core communication layer -> critical
  - Reliable communication in distributed system
  - Decentralized (no brokers, etc.) -> scalable
- Centrally managed middleware services
  - Directory/Naming, RBAC, Proxies, DIP Gateways, …
- Access syntax -> Device-Property model
  - Device: addressable IO point (e.g. channel, slot, module)
  - Device: e.g. LHC.BPM.P5.B1, LHC.BPM.P5.B2
  - Property: exposed operation or method
  - Property: e.g. Acquisition, Status, Alarm, Setting
- Widely deployed for all CERN accelerators
  - Used in all Eqp. groups (3 deps: BE, EN, TE)

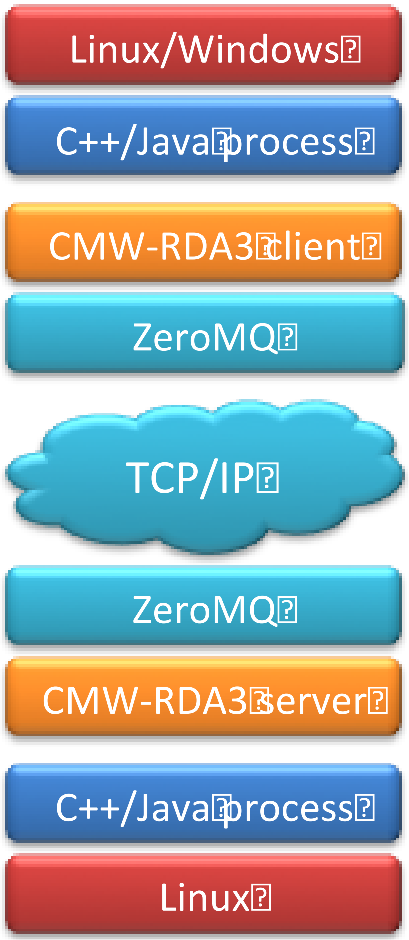


CMW - A bit of history

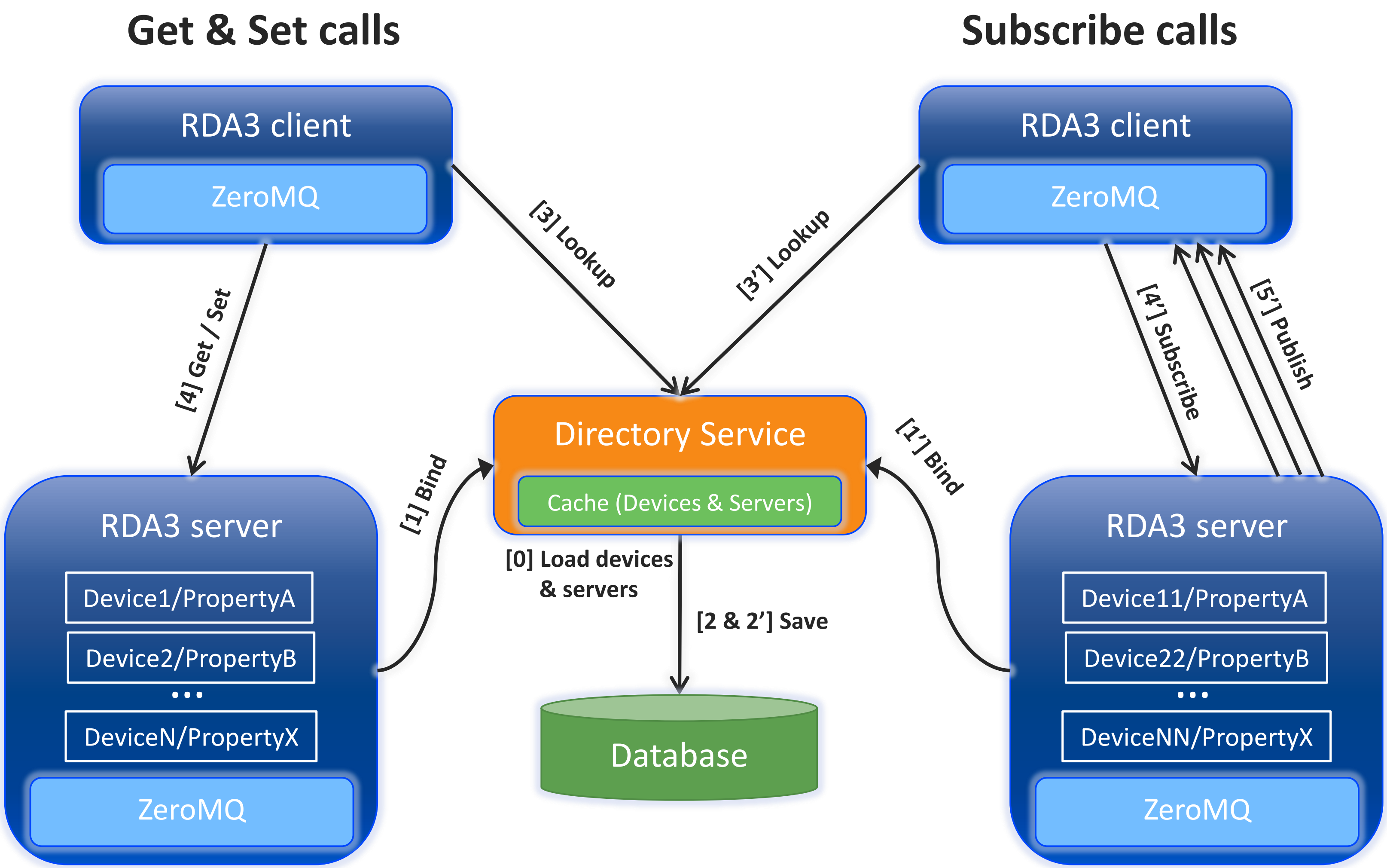
- 1988 – RPC-based middleware
- 2002 – CORBA-based version of RDA (**R**emote **D**evice **A**ccess)
- Problems observed with CORBA:
  - Technical obsolescence & shrinking community
  - Lack of asynchronous transport & comm. patterns (only RPC)
  - Lack of handling of “slow clients” (had to implement workarounds)
  - Poor scalability for subscriptions & for many clients (>200)
- 2011 – review & selection of ZeroMQ as replacement for CORBA
  - See paper: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/frbhmult05.pdf (or: http://cern.ch/go/G9RC)
- 2012-2013 – design & implementation of RDA3 (based on ZeroMQ)
- Spring 2014 – operational deployment of RDA3

RDA3 – CMW core communication library

- Uses **ZeroMQ** for low-level networking
- Built upon lock-free, async (event driven) architecture
- Operations/Calls: *Get, Set, Subscribe*
  - All communication implicitly asynchronous
- Public API for developing clients & servers
- Provided for several platforms
  - C++ (C++98) & Java (Java7)
  - Dependencies: ZeroMQ, Boost (C++)
  - Linux (32/64-bit: SLC5, SLC6), Windows (64-bit)
- Integrated with several frameworks
  - FESA, FGCD, WinCCOA, LabVIEW, C2MON
- Provides comprehensive diagnostics
  - Admin GUI, command line tools, Kibana reports
- Exportable -> used in GSI



Data flow for RDA3 calls



Example – sync Get call

```
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
…
…
// First create ClientService and keep it for all further communication
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();

// Get AccessPoint by providing device name & property name
AccessPoint & accessPoint = client->getAccessPoint(”LHC.BPM.P5.B1", ”Acquisition");

// Perform sync GET call
auto_ptr<AcquiredData> acqData = accessPoint.get();

// The resulting data is obtained by calling getData() method on the AcquiredData instance
const Data & data = acqData->getData();
cout << data.toString() << endl;

// Data -> supports scalars, arrays of scalars(1D, 2D, N-D), structures (nested Data)

// Additionally AcquiredData contains also AcquiredContext, which provides meta-data about the call
const AcquiredContext & acqContext = acqData->getContext();
cout << acqContext.toString() << endl;
```

Example – async Get call with callback

```
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
class Callback : public AsyncGetCallback
{
public:
  void requestCompleted(const RequestHandle & request, auto_ptr<AcquiredData> acqData)
  {  cout << acqData->getData().toString() << endl;  }

  void requestFailed(const RequestHandle & request, auto_ptr<RdaException> exception)
  {  cout << exception->what() << endl;  }
};

auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint(”LHC.BPM.P5.B1", ”Acquisition");

// Perform async GET call with callback
accessPoint.getAsync(AsyncGetCallbackSharedPtr(new Callback()));

// Continue while GET is being performed
...
...
```

Example – async Get call with future

```
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
...
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint(”LHC.BPM.P5.B1", ”Acquisition");

// Perform async GET call with future
RdaGetFutureSharedPtr future = accessPoint.getAsync();


...
// Continue processing while GET is being performed
...
cout << “Is GET completed: ” << future->isDone() << endl;
...

// Perform blocking call until GET data is ready
auto_ptr<AcquiredData> data = future->get();

cout << data->getData().toString() << endl;
```

Example code – sync Set call

```
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
...
// First create ClientService and keep it for all further communication
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();

// Get AccessPoint by providing device name & property name
AccessPoint & accessPoint = client->getAccessPoint(”LHC.BPM.P5.B1", ”Setting");

// Prepare data to be sent to server
// Data -> supports scalars, arrays of scalars(1D, 2D, N-D), structures (nested Data)
auto_ptr<Data> data = DataFactory::createData();
data->append("value", 10);
data->append(”gain", 5.1234);

// Perform sync SET call
accessPoint.set(data);
```

→Analogically async Set versions with callback & future

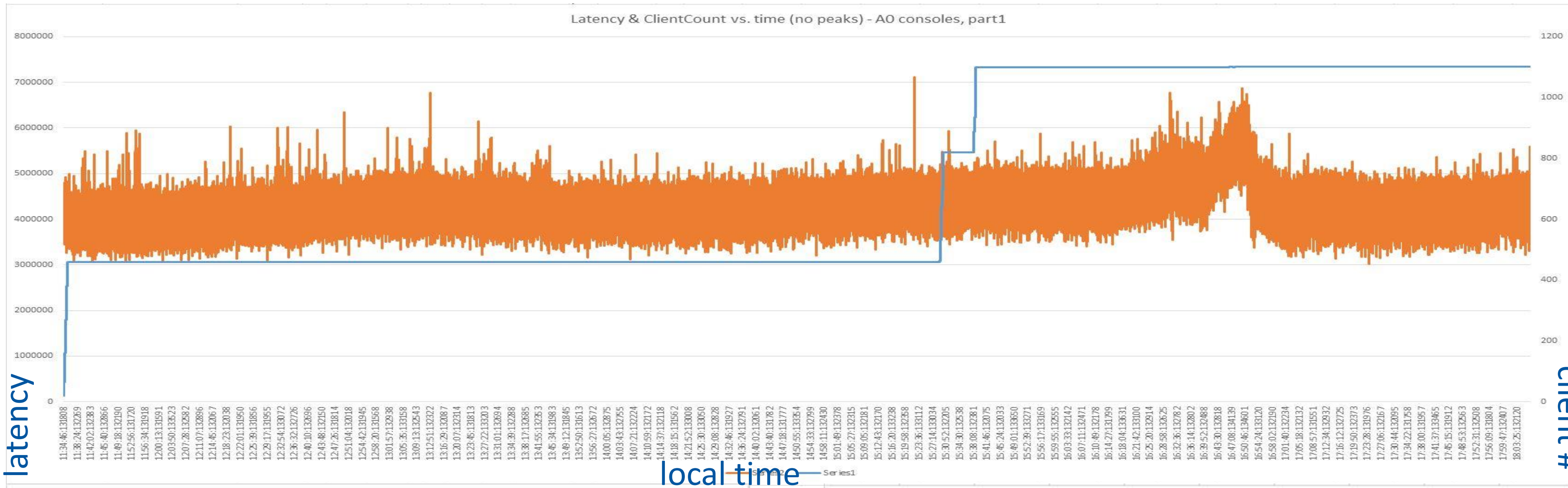
Example – Subscribe call

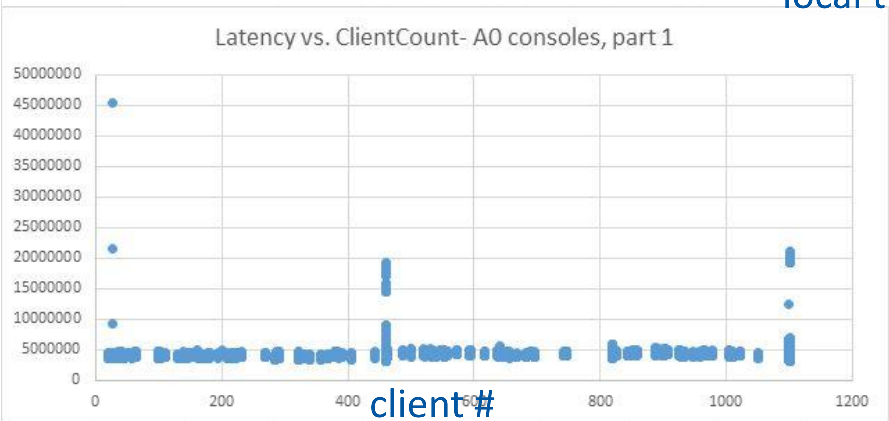
```
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
...
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint(”LHC.BPM.P5.B1", ”Status");

// Perform SUBSCRIBE call
SubscriptionQueueSharedPtr queue = accessPoint.subscribe();
...
// Continue processing while subscription is being established
...
// Blocking wait for a single notification with a timeout of 5000 ms
std::auto_ptr<Notification> notification = queue->poll(5000);  // also with no timeout: poll()
cout << “Received notifications count: ” << queue->getQueueSize() << endl;

// Process the received data
auto_ptr<AcquiredData> acqData = notification->get();
cout << acqData->getData().toString() << endl;
```

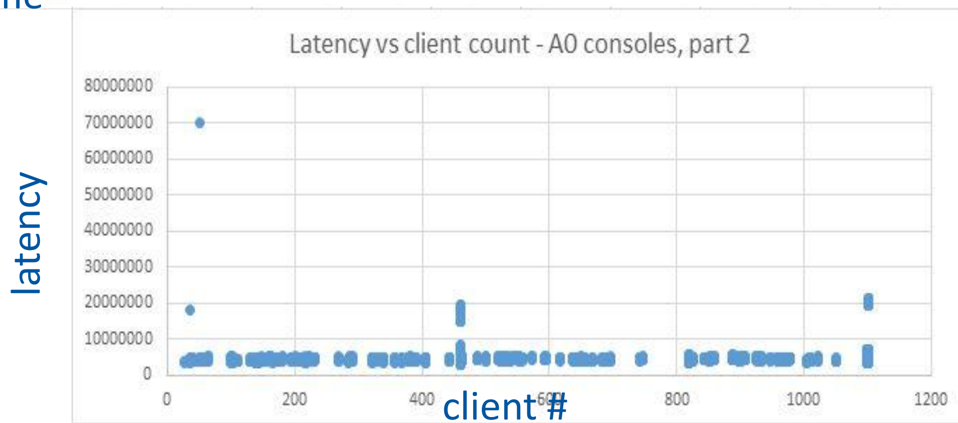
→Analogically Subscribe version with callback

Latency stability measured by BE-CO Timing team

Setup: 500B message payload, CCC machines, 1 server/publisher, 1-1100 clients/subscribers

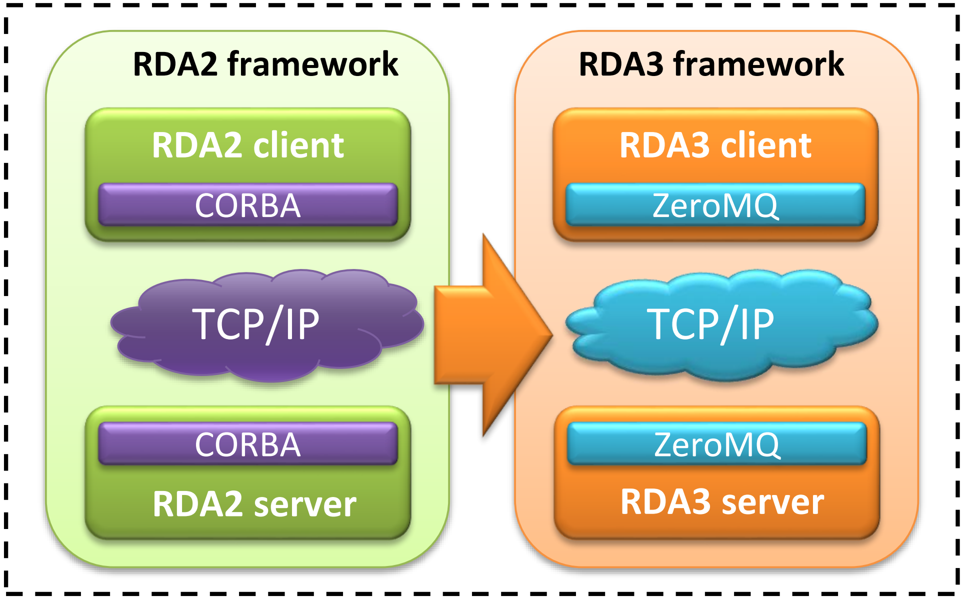




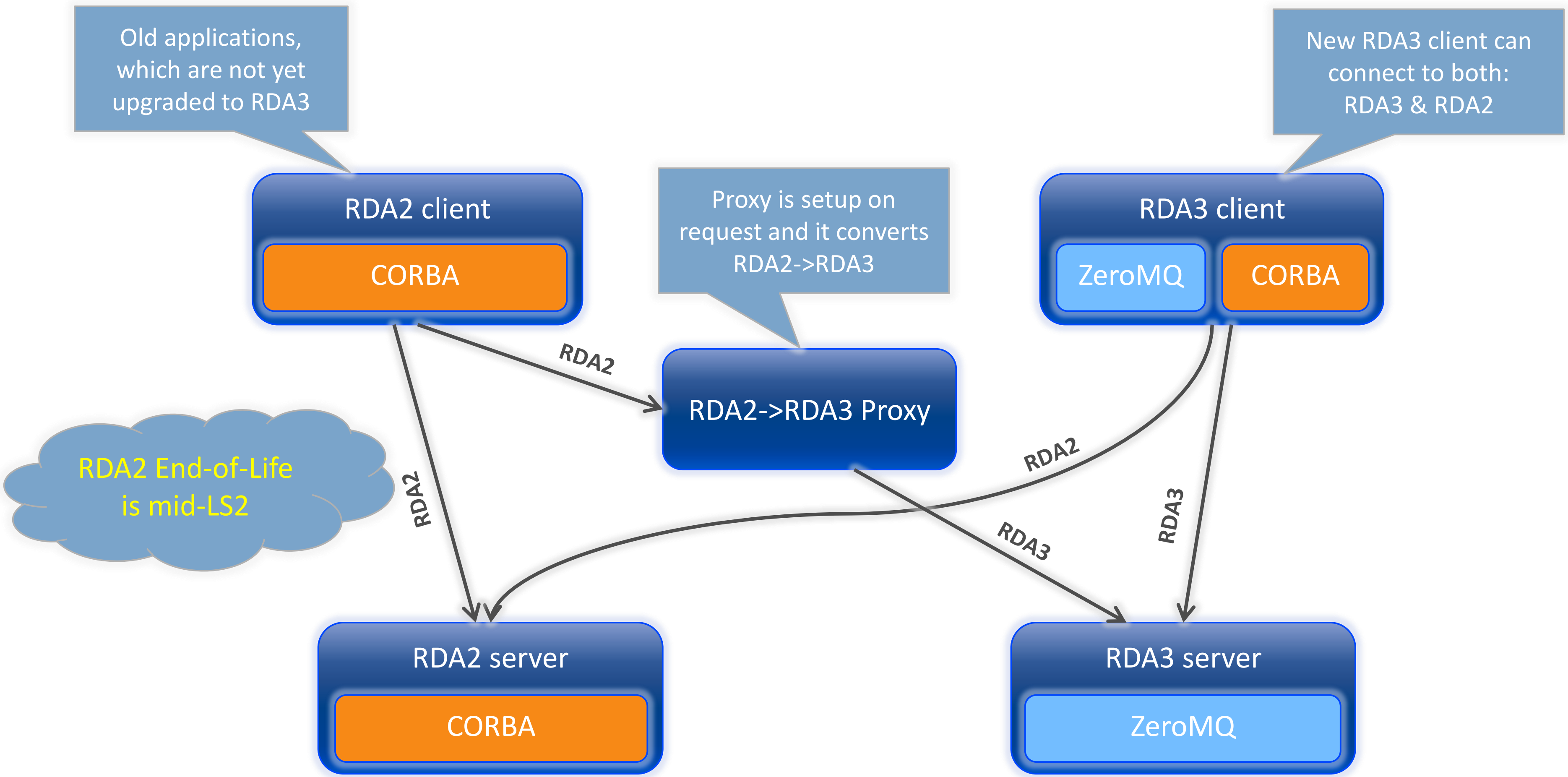


Observations from operational use of RDA3

- Great performance for subscriptions (thanks to async transport & internal batching)
- Good scalability for many clients (>1000)
- No more problem of slow clients (solved properly thanks to async transport)
- Much smaller footprint (CPU & memory), mainly thanks to zero-copy
- ZeroMQ - no surprises & very reliable!



CMW ”mixed” infrastructure today



Future plans

- Q4 2016 – provide REST API to access RDA devices
  - Via dedicated REST->RDA gateways
  - Requested by web developers & scripting community
- Q1/Q2 2017 – provide Python binding for RDA
  - On top of REST API, no dependency on RDA
- Q3 2017 – new major version of Directory Service

Resources

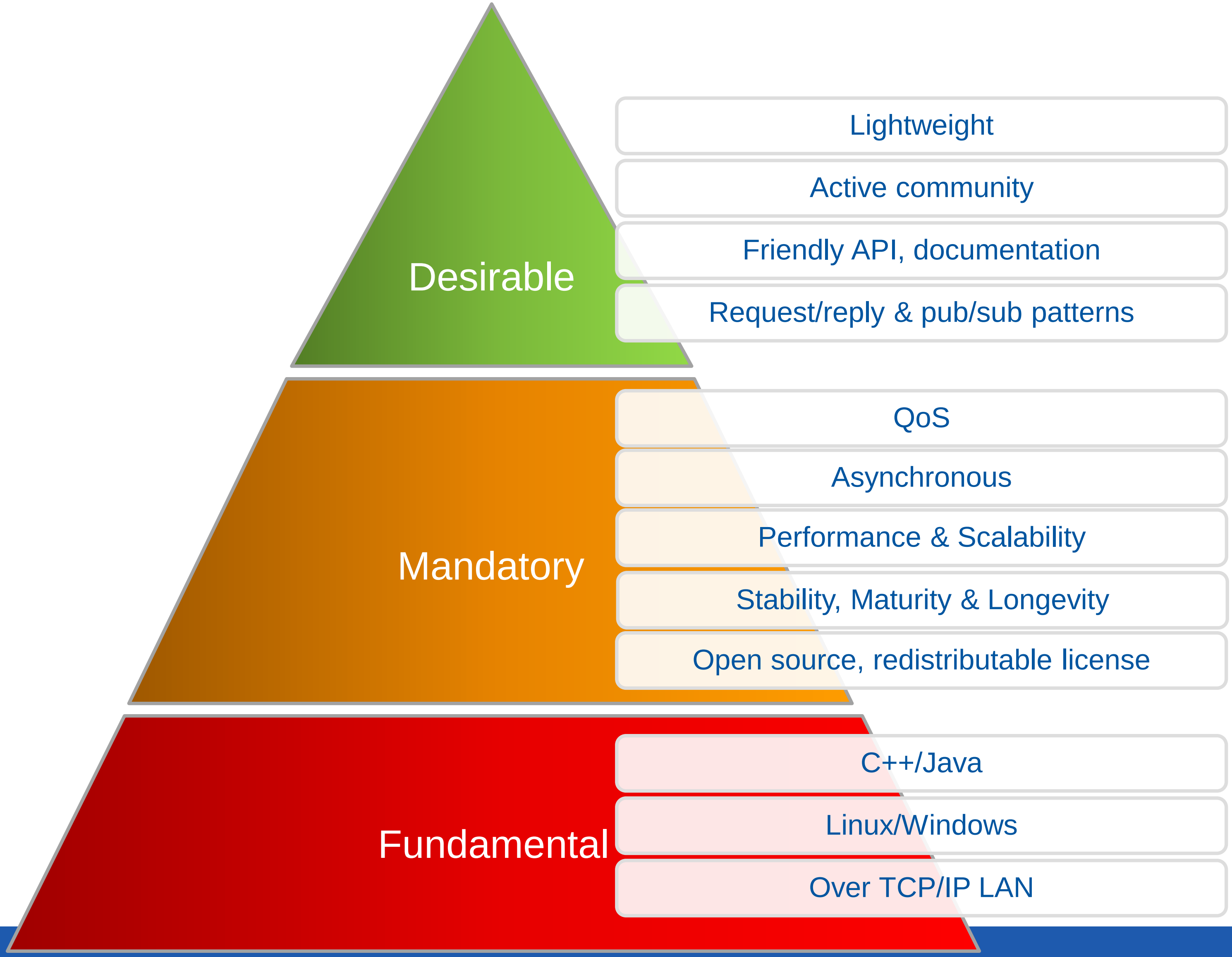
- CMW Wikis: https://wikis.cern.ch/display/MW/
- RDA3 User Guide: https://wikis.cern.ch/display/MW/RDA3+User+Guide
- RDA3 Java: https://svnweb.cern.ch/cern/wsvn/acc-co/trunk/cmw/cmw-rda3/
- RDA3 C++: https://svnweb.cern.ch/cern/wsvn/acc-co/trunk/cmw/cmw-rda3-cpp/
- Middleware Review: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/frbhmult05.pdf (or: http://cern.ch/go/G9RC)
- Emails:
  - `cmw-support@cern.ch` (questions, requests, etc.)
  - `cmw-news@cern.ch` (news from MW team, subscribe via e-groups)
  - `Wojciech.Sliwinski@cern.ch` (or: `w.s@cern.ch`)

Conclusions

- Middleware is complex & challenging domain
- Many solutions exist -> understand your system and it's requirements before picking-up a product
- Modular design & generic, narrow public API should allow for future evolution
- Do you need to stay backward-compatible?
- What about security? Yes, it has to be provided
- RDA3 is reliable & generic –> can be used elsewhere
- ZeroMQ proved to be stable & very efficient

Backup slides

CERN Middleware Requirements



How did we evaluate –> our criteria

Appearance

- **Creators**
  - specification
  - documentation
- **Users**
  - forums
  - bug reports
- **Internet**

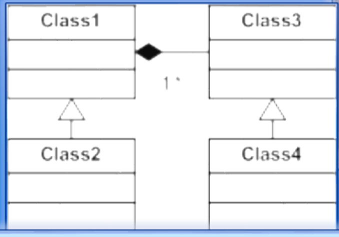
Simple usage

- **Download**
  - licensing
- **Compile**
  - LynxOS & gcc 2.95
- **Run examples**

Testing

- Communication patterns
- Performance
- QoS
- Exceptional situations

CRITERIA



API, look & feel, documentation

resources, binary size, memory

Community, maturity

Communications patterns

QoS

performance

Evaluated middleware products

All opinions are based only on our knowledge and evaluation. Each of the products, depending on the requirements, may constitute a good solution.

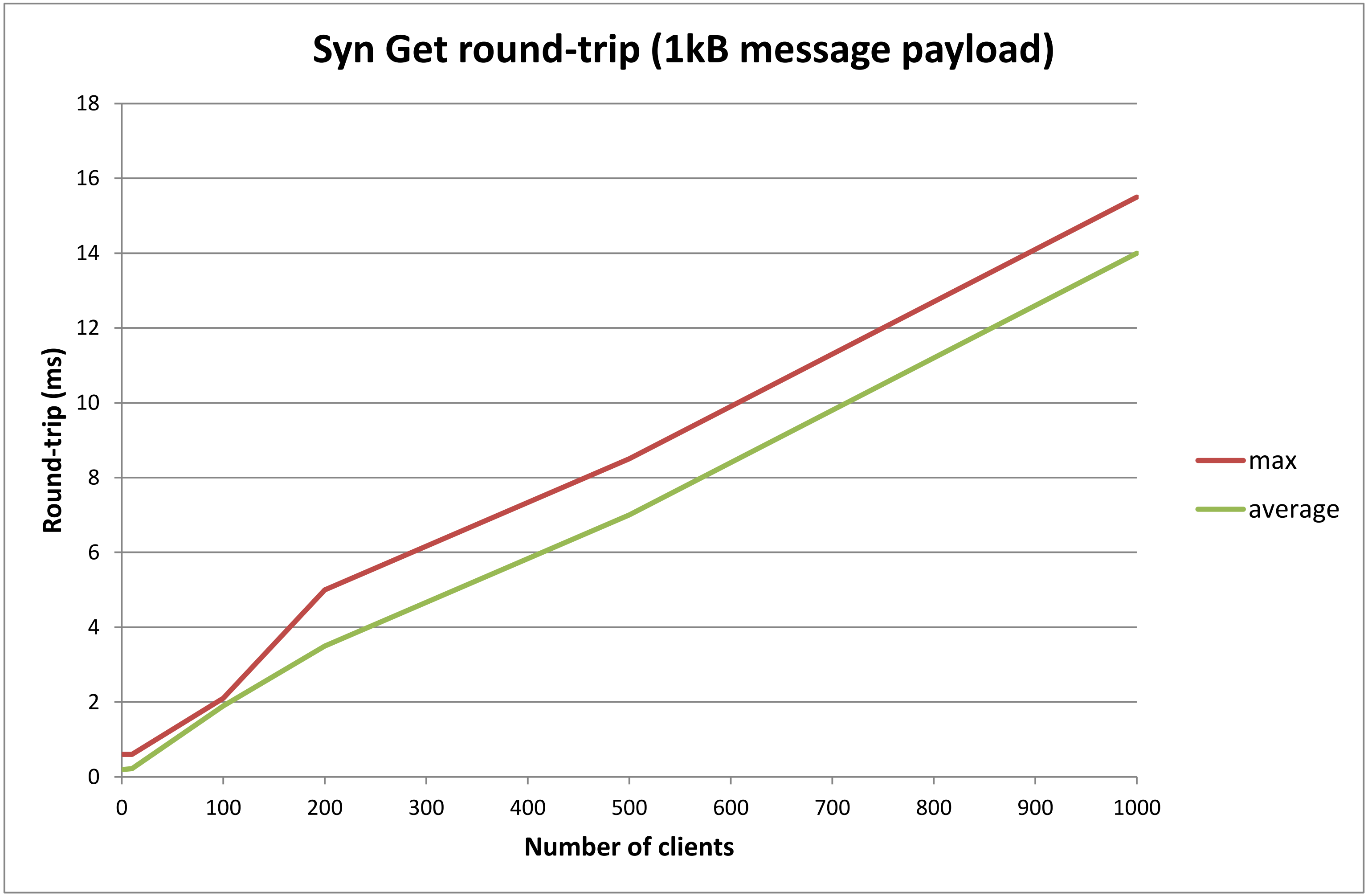
- OpenAMQ
- CoreDX
- RTI DDS
- QPid
- ZeroMQ
- OpenSpliceDDS
- RabbitMQ
- Ice
- YAMI
- omniORB
- JacORB
- MQtt RSMB
- Thrift
- Mosquito

Products comparison (according to the criteria)

	Sync, async & msg patterns	QoS	Dependencies & memory f-p	Performance	Look & feel, API, docs	Community & maturity	Score
ZeroMQ	✓	✓	✓	✓	✓	✓	6
Ice	✓	✓	✗	✓	✓	✓	5
YAMI4	✓	✓	✓	✗	✓	✗	4
RTI	✗	✓	✗	✓	✗	✓	3
Qpid	✗	✓	✗	✗	✓	✓	3
CORBA	✗	✓	✗	✓	✗	✗	2
Thrift	✗	✗	✓	✓	✗	✗	2

RDA3 Java – Sync Get round-trip time

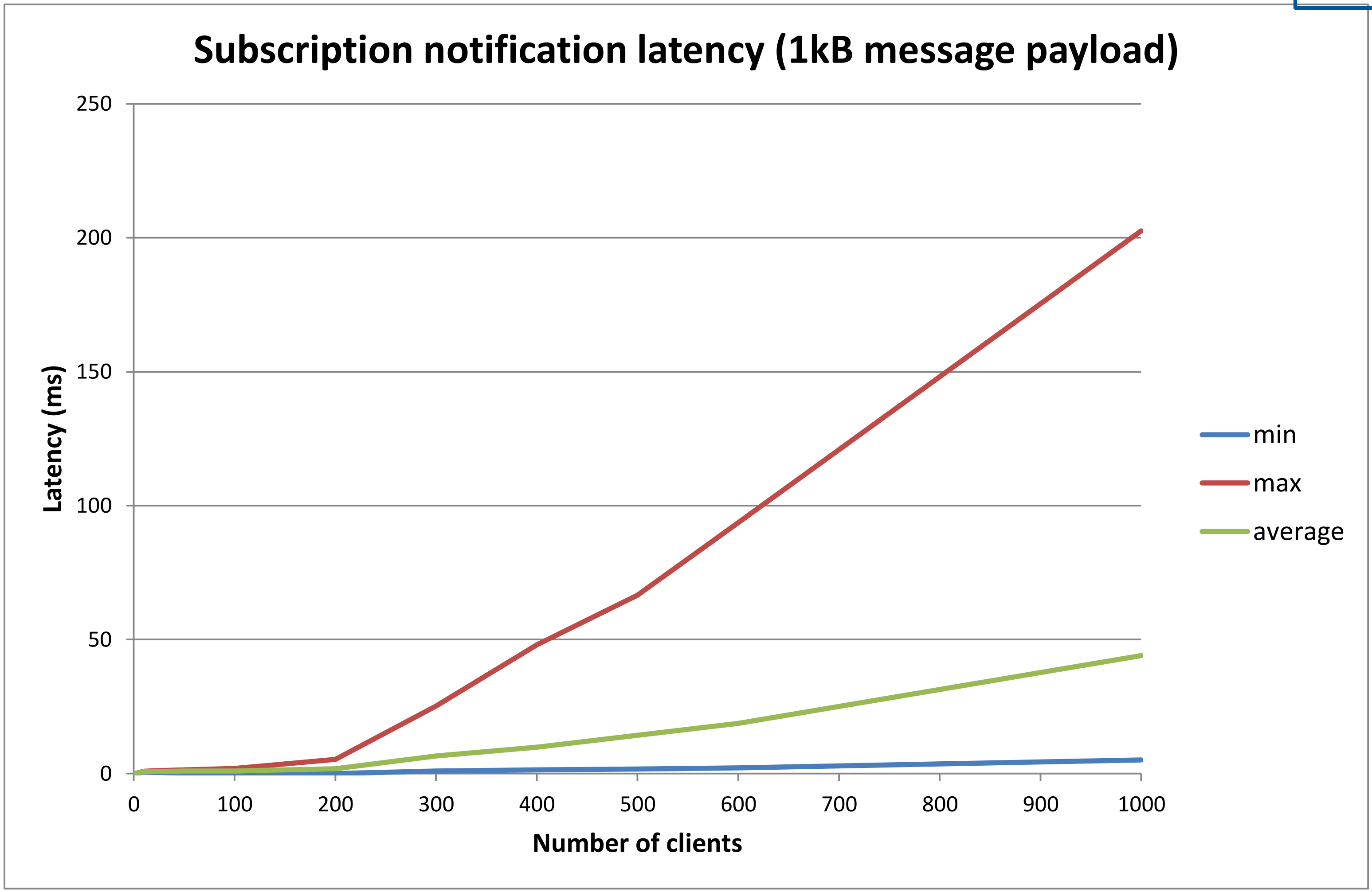
June'13



Test setup: 1kB message payload, cs-ccr-* machines, 1 server host & 10 client hosts

RDA3 Java – Subscription notification latency

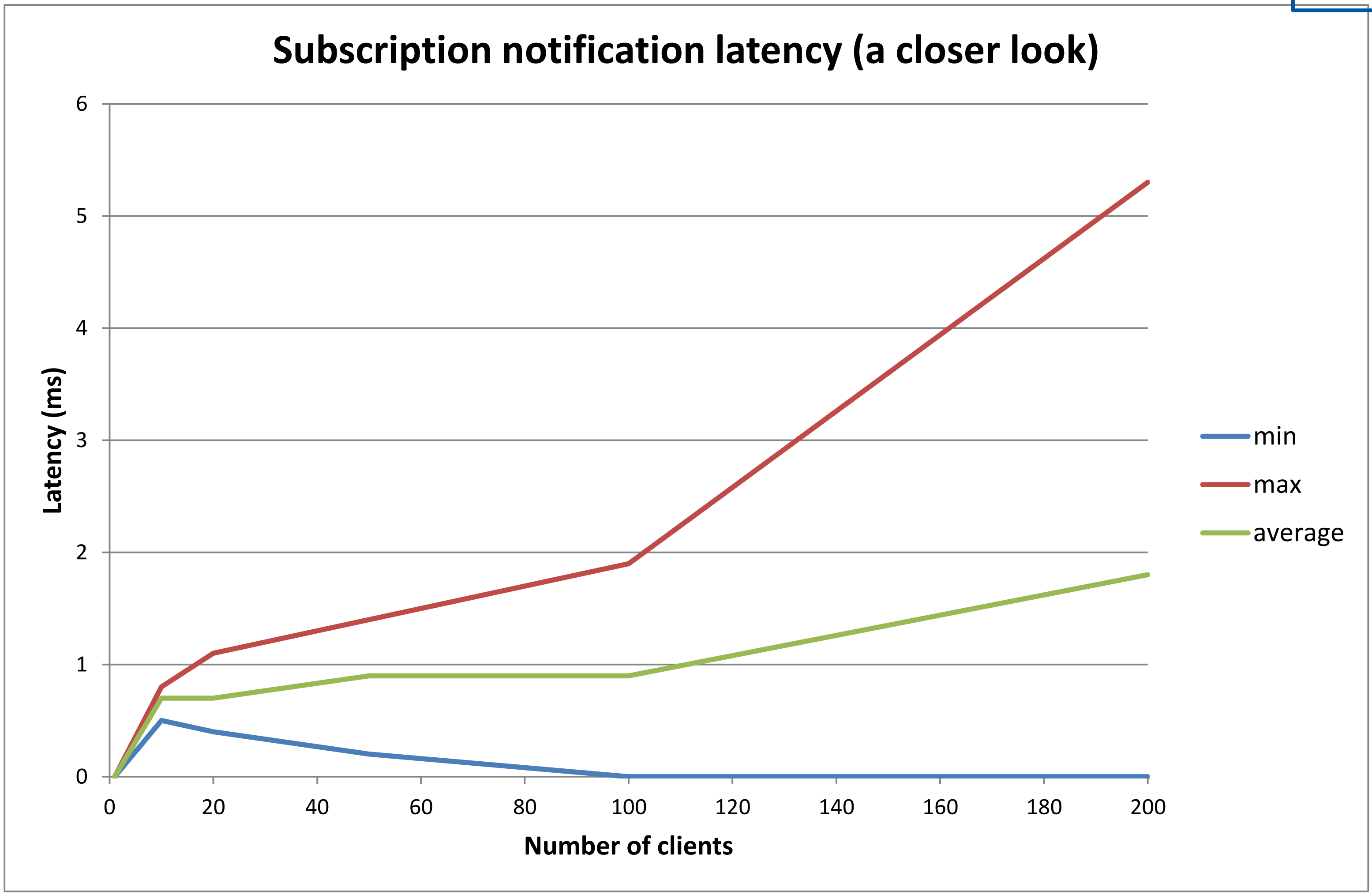
June'13



Test setup: 1kB message payload, cs-ccr-* machines, 1 server host & 10 client hosts

RDA3 Java – Subscription notification latency

June'13



Test setup: 1kB message payload, cs-ccr-* machines, 1 server host & 10 client hosts