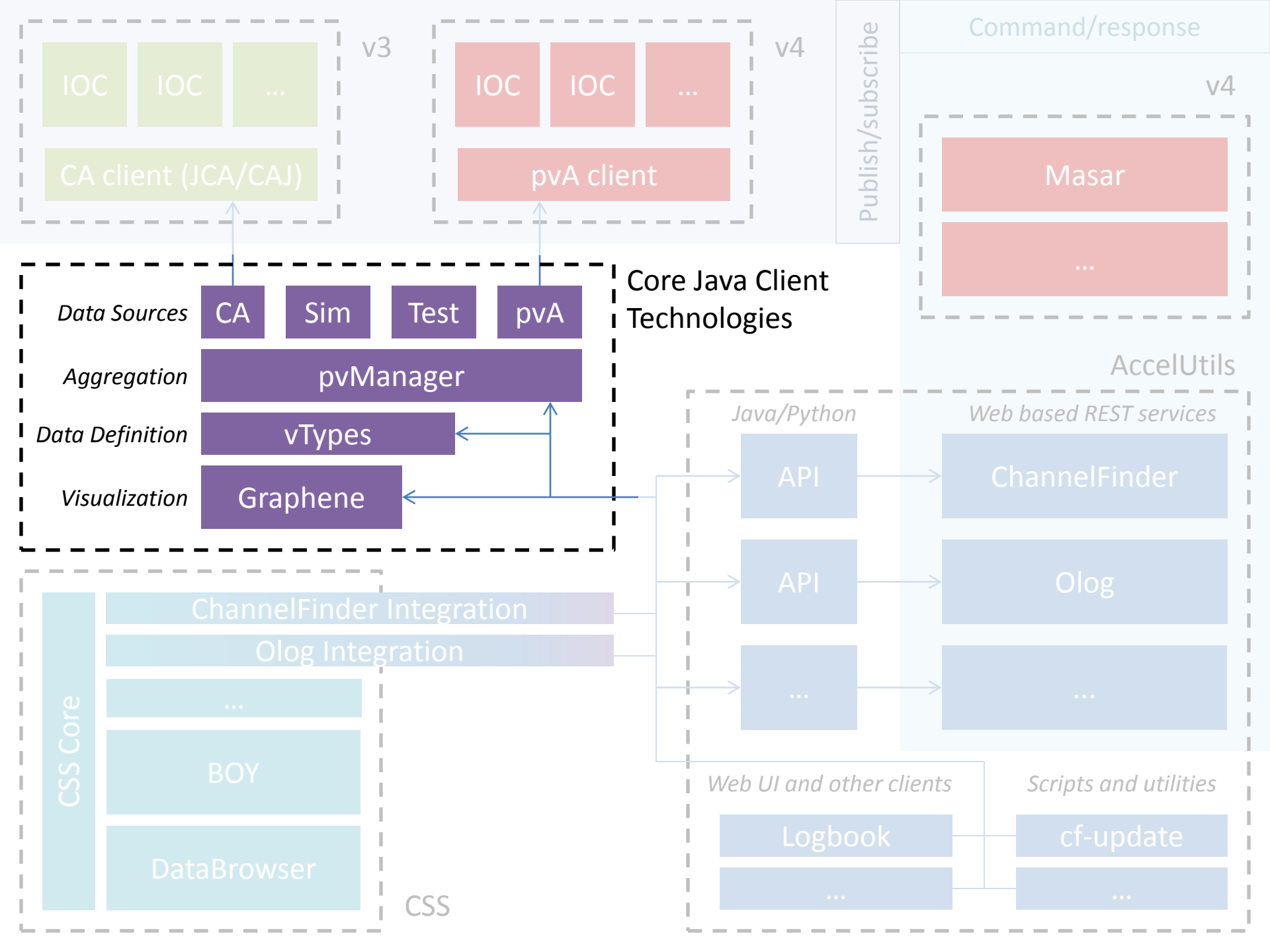
Core Java Client Technologies

Gabriele Carcassi - BNL


v3
IOC
IOC
...
CA client (JCA/CAJ)
v4
IOC
IOC
...
pvA client
Publish/subscribe
Command/response
v4
Masar
...
Core Java Client Technologies
Data Sources
CA
Sim
Test
pvA
Aggregation
pvManager
Data Definition
vTypes
Visualization
Graphene
AccelUtils
Java/Python
Web based REST services
API
ChannelFinder
API
Olog
...
...
CSS Core
ChannelFinder Integration
Olog Integration
...
BOY
DataBrowser
CSS
Web UI and other clients
Scripts and utilities
Logbook
cf-update
...
...

pvManager

- Provide logic required by well-behaved application
  - Queuing or caching
  - Rate decoupling between subsystems (typically CA and UI, but also CA and anything else)
  - Rate limiting (notification rate capped)
  - Rate throttling (notification rate adapts, skips instead of building up lag)
  - Shared connections (similar requests on a single monitor)
  - Ability to offload work on a shared pool
- You still need to understand what all of this is
- You just don’t have to implement it

pvManager sightings

- CSS related: Eric Berryman (MSU), Xihui Chen (ORNL), Marcus Michalsky (PTB)
- Bastian Knerr and Jan Hatje (DESY) implemented an archive engine using pvManager
  - Contributed back some code
- Murali Shankar (SLAC) investigating use for his archive engine

pvManager examples

```
// Read channel "channelName" up to every 100 ms
PVReader<Object> pvReader =
PVManager.read(channel("channelName")).every(ms(100));
pvReader.addPVReaderListener(...)
```

```
// Read channel "channelName" up to every 100 ms, and get all
// the new values from the last notification.
PVReader<List<Object>> pvReader =
PVManager.read(newValuesOf(channel("channelName"))).every(ms(100));
```

```
// Read a map with the channels named "one", "two" and "three"
PVReader<Map<String, Object>> pvReader =
    PVManager.read(mapOf(
        latestValueOf(channels("one", "two", "three"))))
    .every(ms(100));
```

Syntax style: functional, fluent API, internal domain specific language (iDSL)

pvManager

- Push the logic out of the UI
  - Added for ChannelFinder PVTable by property

```
// Create a table with 2 columns, pv name and value, and 3 rows
List<String> names = Arrays.asList("one", "two", "trhee");
PVReader<VTable> pvReader = PVManager.read(vTable(
    column("Names", vStringConstants(names)),
    column("Values", latestValueOf(channels(names)))))
    .every(ms(100));
```

  - Trivial to unit test the aggregation logic outside of the UI with mock data
  - Logic is now available to use without CSS or ChannelFinder

pvManager

- Push the computation out of the UI
  - Added for histogram

```
// Create a table with 2 columns, pv name and value, and 3 rows
PVReader<VImage> pvReader =
PVManager.read(histogramOf(vDouble("myChannel")))
    .every(hz(50));
```

  - Histogram is computed and plotted on background thread
  - If UI thread is loaded, frames are automatically skipped
  - Available to use without CSS/Eclipse/SWT

The search for graphs

- 18 months looking for graphics package
  - Suitable for concurrency
  - Could be used within or without UI
  - Performance suitable for real-time
  - Reasonable footprint
- Closest we could find jHepWorks
  - Wraps around different 3rd party libraries with a homogenous dataset/plot interface… BUT
  - Each plot would bring in MBs of dependencies
  - Could not be computed out of the UI (just using their classes outside their framework was not easy)
  - Dataset are concrete classes (not interfaces)
  - No atomic data update
  - For a histogram, 600x400 was 5ms per draw
  - Quality of the plot not great

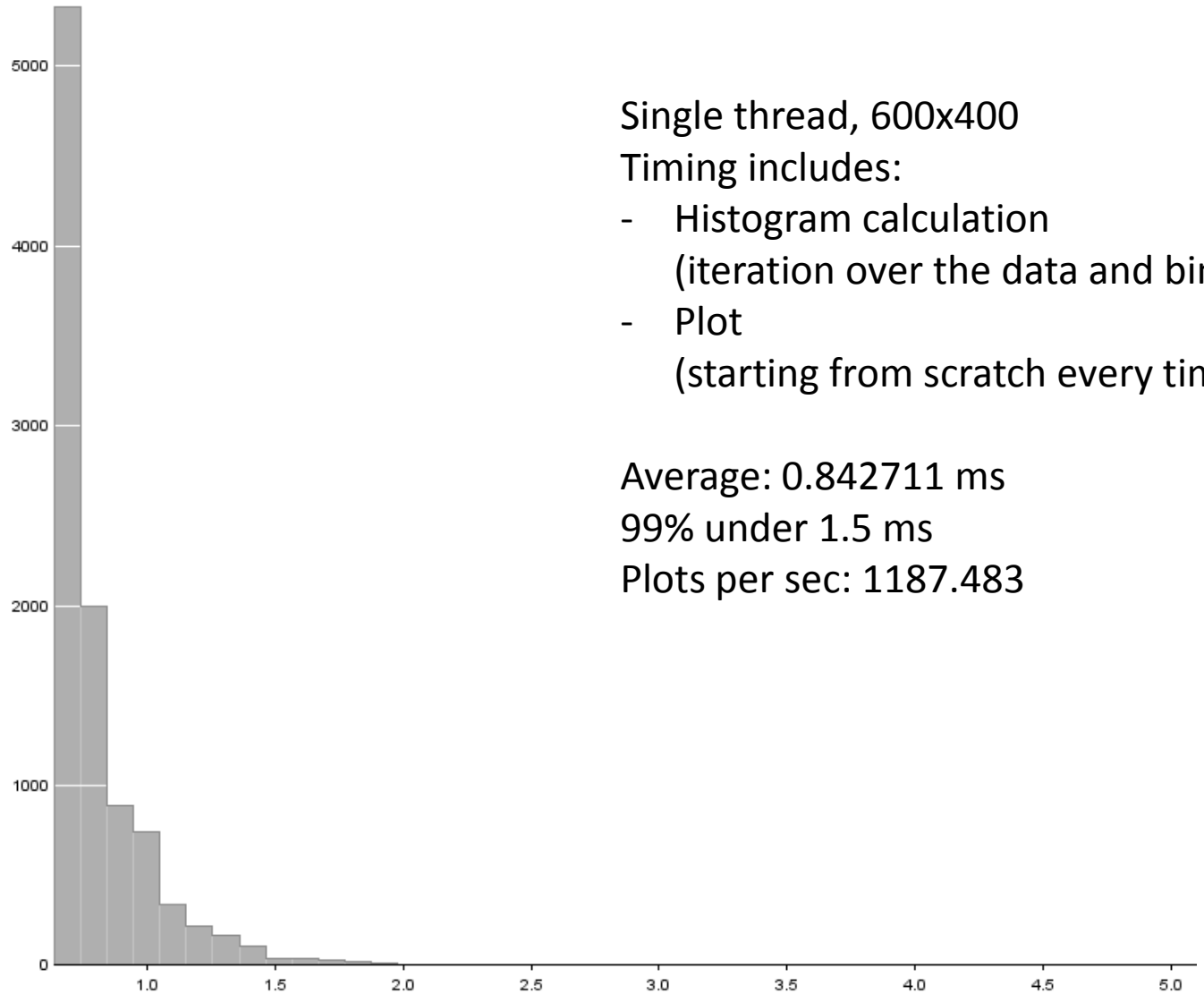
1:51 minutes

CHARTING PROTOTYPE

Histogram (alpha)

PV Name:

Histogram performance histogram



Single thread, 600x400
Timing includes:

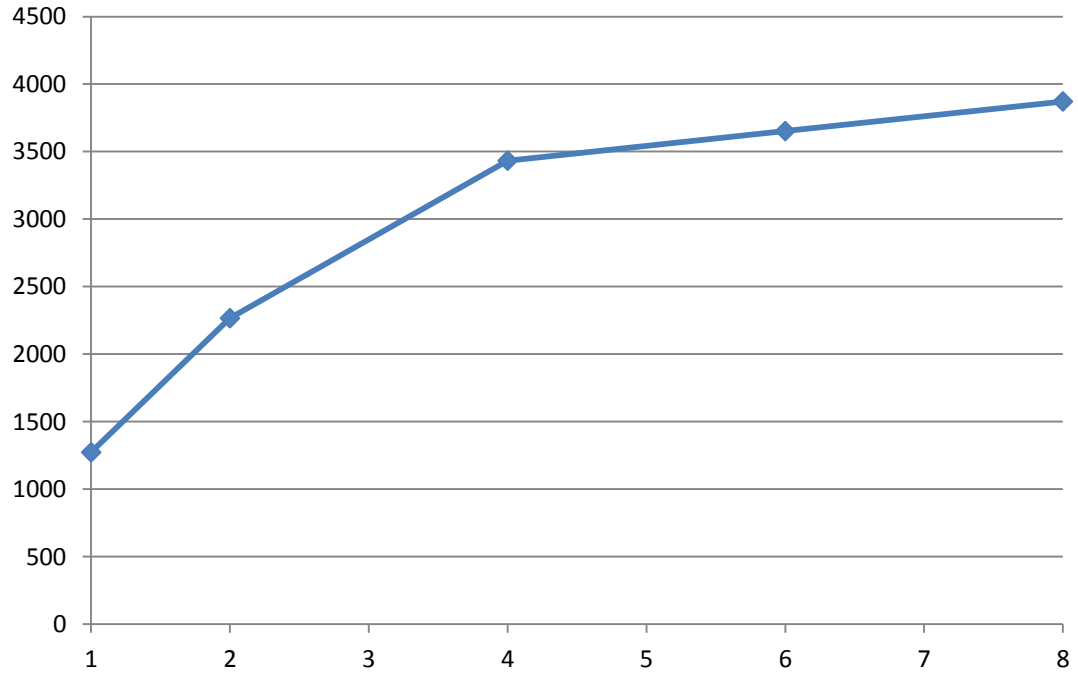
- Histogram calculation
  (iteration over the data and binning)
- Plot
  (starting from scratch every time)

Average: 0.842711 ms
99% under 1.5 ms
Plots per sec: 1187.483

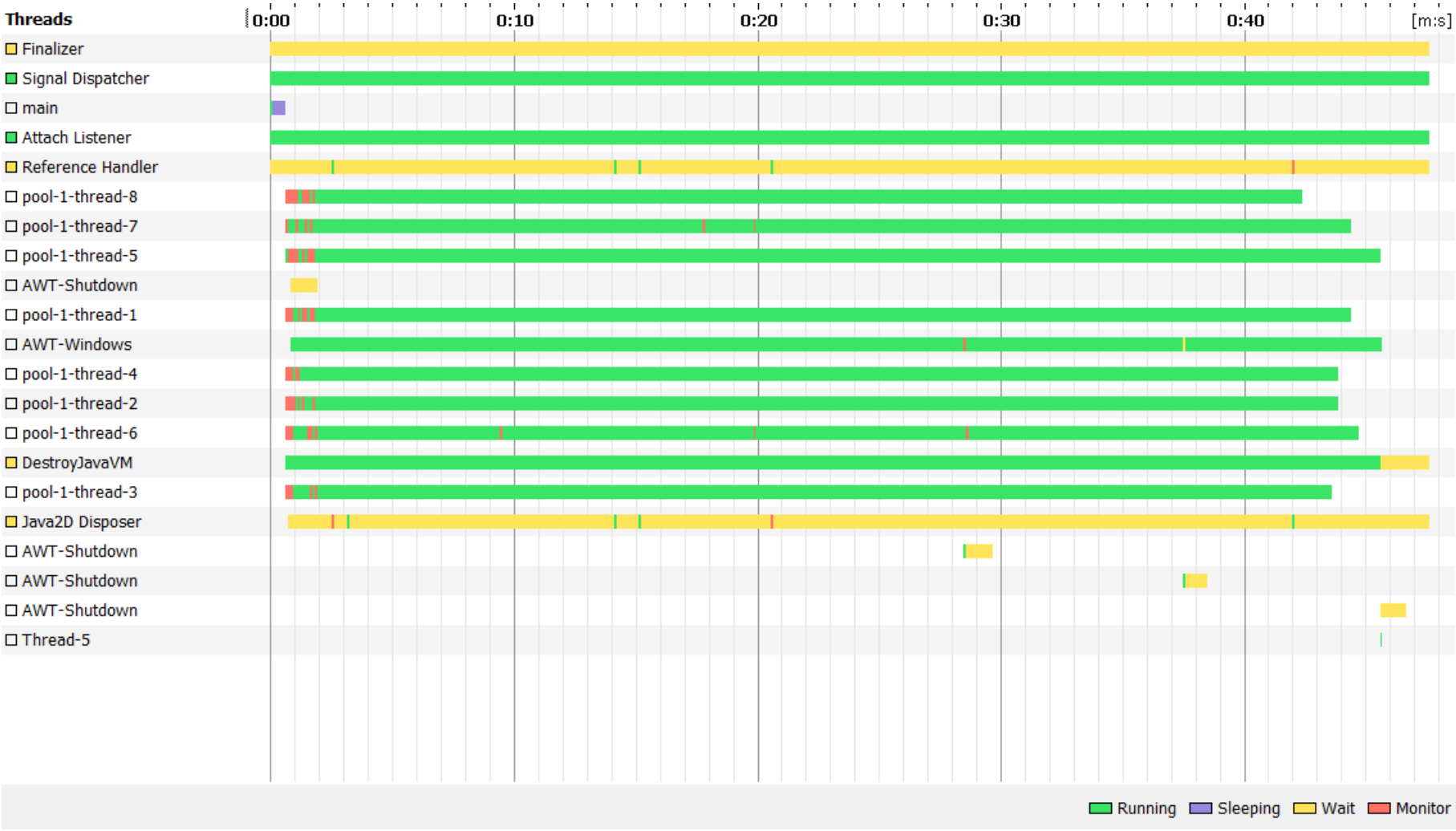
Scalability on multiple cores

N threads	1	2	4	6	8
Average time (ms)	0.785532	0.882468	1.16518	1.642851	2.066334
N plot per sec	1273.022	2266.372	3432.947	3652.188	3871.591

Intel Core i7-840
Quad core
1.87 GHz



Lack of locking



Graphene

- Interfaces for datasets
- Rendering without UI
  - Plot.draw(Graphics2D graphics, Dataset dataset)
- Changes are thread-safe and atomic
  - Dataset.update(DatasetUpdate change)
  - Once class encapsulate the change, which can be shipped to a different thread
- “Publishable quality” (from Eric Berryman)
  - Taking ideas from Edward Tufte and other libraries
- We understand that it’s a lot of work
  - Couldn’t find alternative that works
  - We don’t have to do it all

pvManager laundry list

- Generalize type support
  - Would better address Desy and v4 requirements
  - Pluggable mechanism to convert datasource payload to client datatype
  - Pluggable mechanism to convert between client datatypes
- Review and finalize write support
- Review error support
  - Some errors are not reported correctly (e.g. does not connect because of max_array_bytes too low)
  - Exceptions are not multi-casted to all readers/writers
- Cache with connection to historic data
- Figure out “pausing”
- Add other operations/transformation
  - FFT, use local timestamp, aggregate to annotated POJOs, …

vTypes laundry list

- Figure out a better way to handle different primitive types, so that client does not have to code to each
- Review use of primitive arrays as make it impossible to have no-copy convertions
- Dynamic type conversion expression
  - `toVType(VDouble.class, channel("mychannel")` instead of `toVDouble(channel("mychannel"))`

Graphene laundry list

- More charts
  - Move waterfall plot out of pvManager
- Make current charts customizable
  - Zone of interest for waterfall
- General purpose pieces
  - ColorScheme
  - InterpolationScheme