goto; conference

Click 'engage' to rate sessions and ask questions

High Throughput Horizontally Scalable Workflow

Daniel Meyer
@MeyerDan

Daniel Meyer
Technical lead
Camunda BPM

Twitter:
@MeyerDan

Blog:
http://long-running.net

What is Camunda BPM?

Camunda BPM is an Open Source platform for Business Process Management and allows you to model and execute BPMN, CMMN and DMN.

References

References in the US: AT&T, Financial Regulatory Authority (FINRA), Sony DADC, Spartasystems, …

Great, but what if?

What if we could build a workflow engine with the following properties:

- **Up to 100x better throughput** (compared to current generation)
- **Horizontally scalable**
- **Seamlessly easy to operate at small and large scale**

Why this Talk?

We need your help to make the right decisions.

We need feedback on these issues and we want to start conversations with potential early adopters.

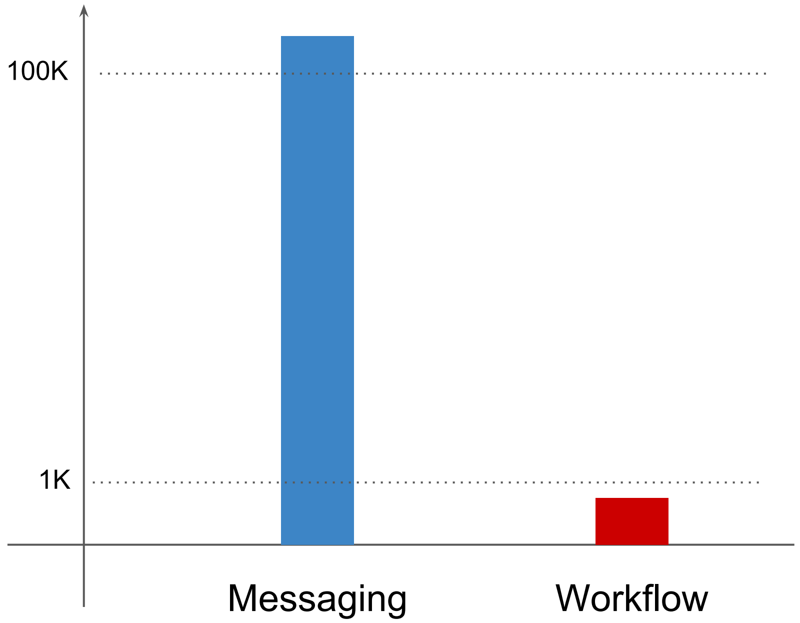
We are an open source project.

1. **Why another Workflow Implementation?**
2. How to design this?

Workflow engines could be **much faster**

Throughput: Messaging vs. Workflow

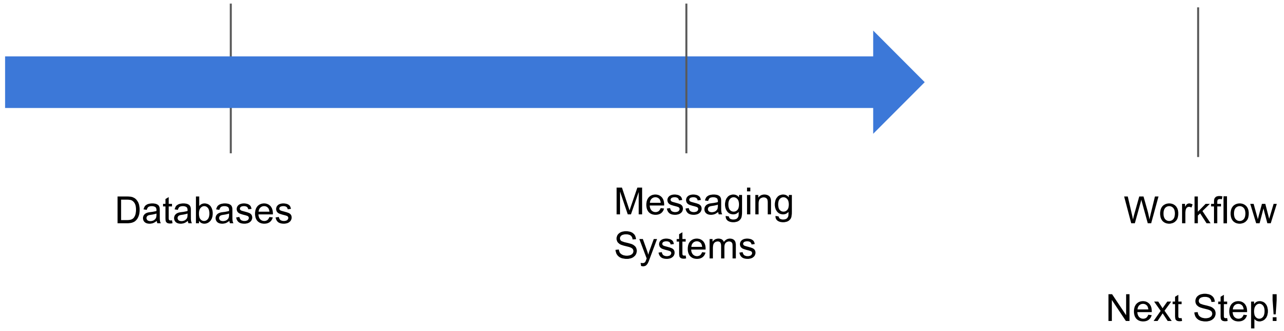
ops / second



persistent messaging systems > 100x better throughput on same hardware

Workflow engines could be **horizontally scalable**

Horizontal Scalability



Performance matters

There is more and more stuff

More and more users produce more and more data and expect increasingly faster and better service.

IoT is coming: 25 billion connected things in 2020 says Gartner.

New, innovative business models will intensify the usage and push data volumes.

BPM can play a key role in this transformation

BPM (Business Process Management) allows Business Analysts, Domain Experts and Developers to collaborate.

Workflow Engines could be much more **modular**

Current Camunda BPM Platform

Message and
Event Correlation

Human Task
Management

History and Audit
Log

Connectors for
Services and
other APIs

Workflow

Repository

Identity
Management

Timers and
Schedules

Business Decision
Management

WENGER
OFFICIAL WORLD SCOUT KNIFE

Modularity

A modular approach allows users

- choose only the components you really need
- scale components individually
- distribute components individually
- use different persistence technology for different components

Yet, it can be simple to use all the components as “one stack”.

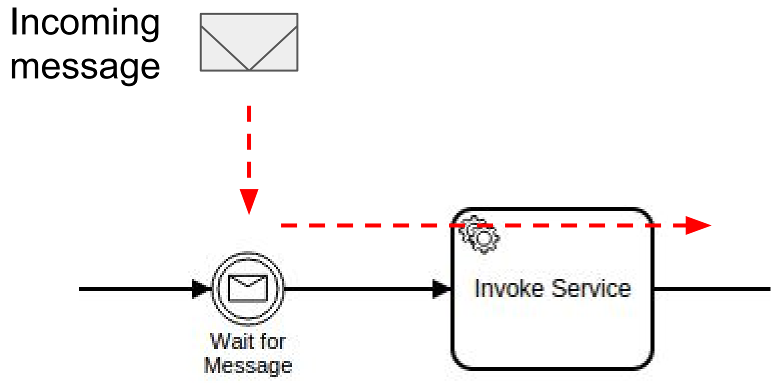
Why yet another Workflow Implementation?

Because
we can do so much better.

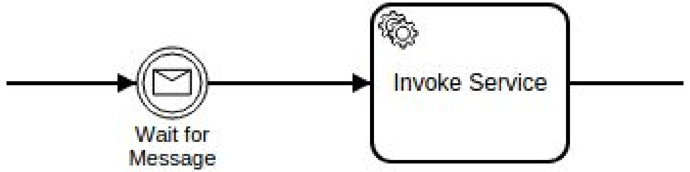
1. Why another Workflow Implementation?
2. **How to design this?**

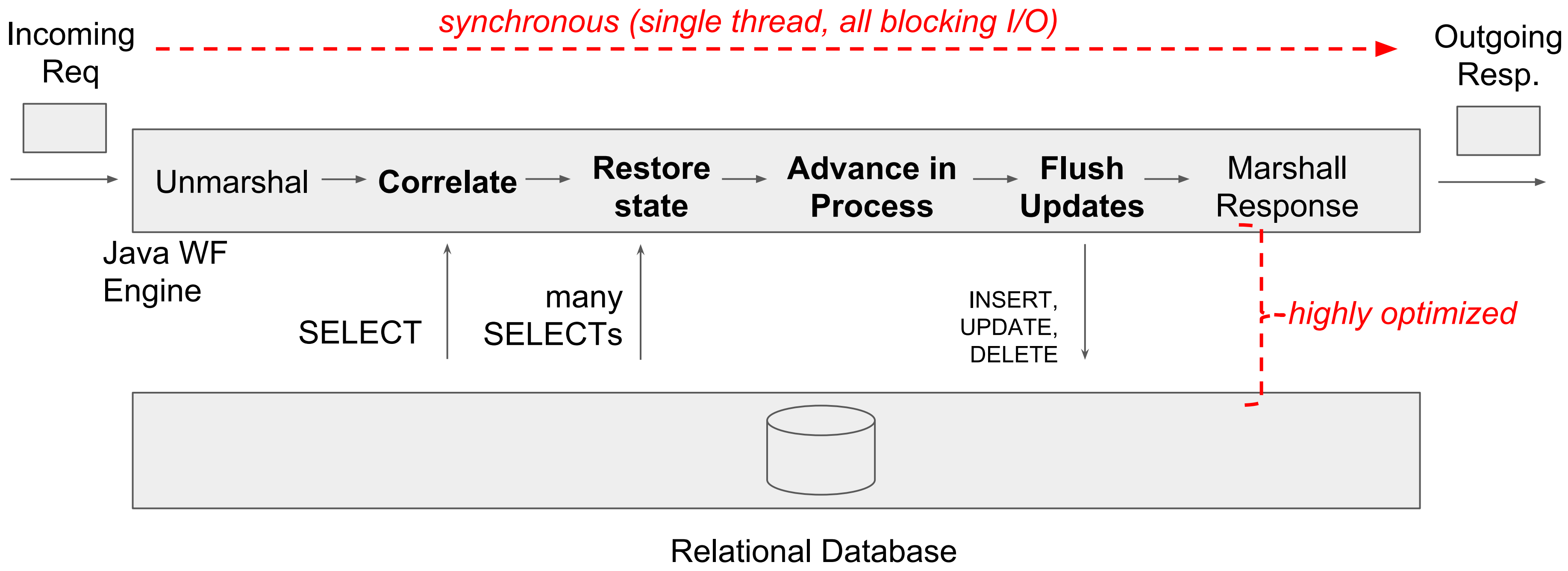
How does Camunda 7 work?

Correlate Message

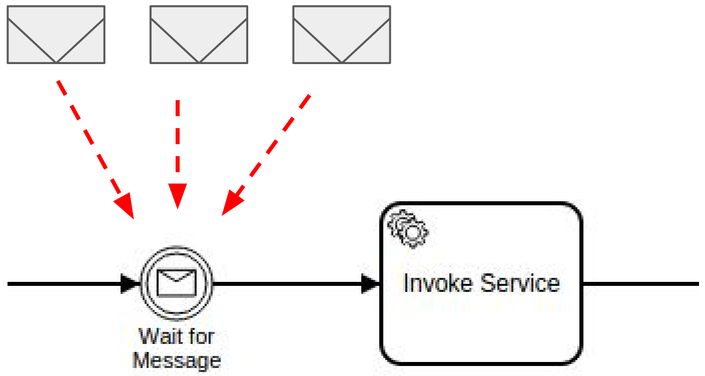


Correlate Message



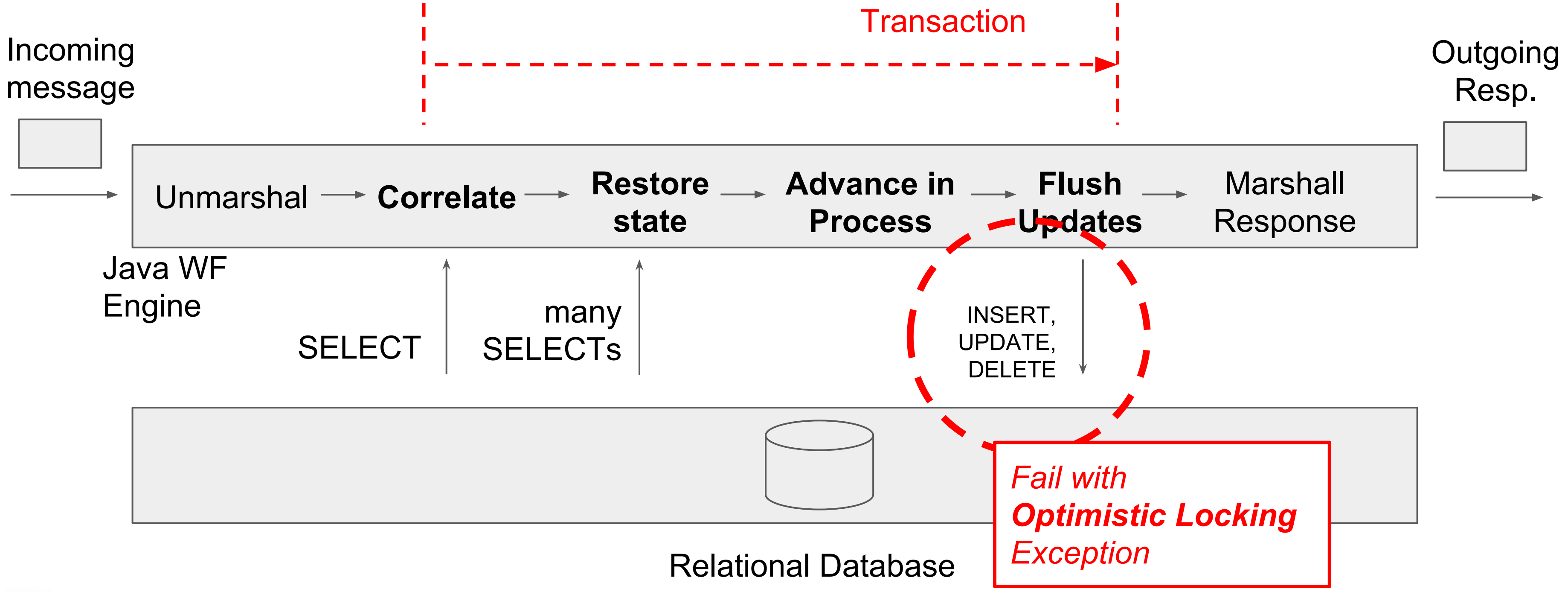


Mutual Exclusion

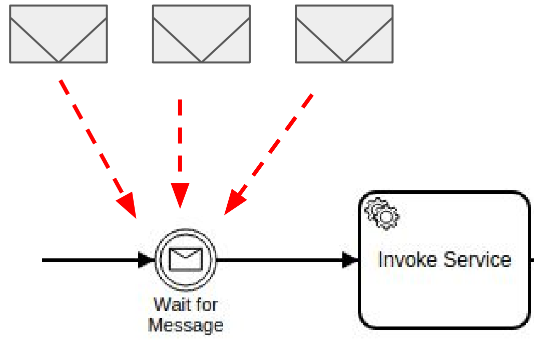


“Racing incoming Signals”

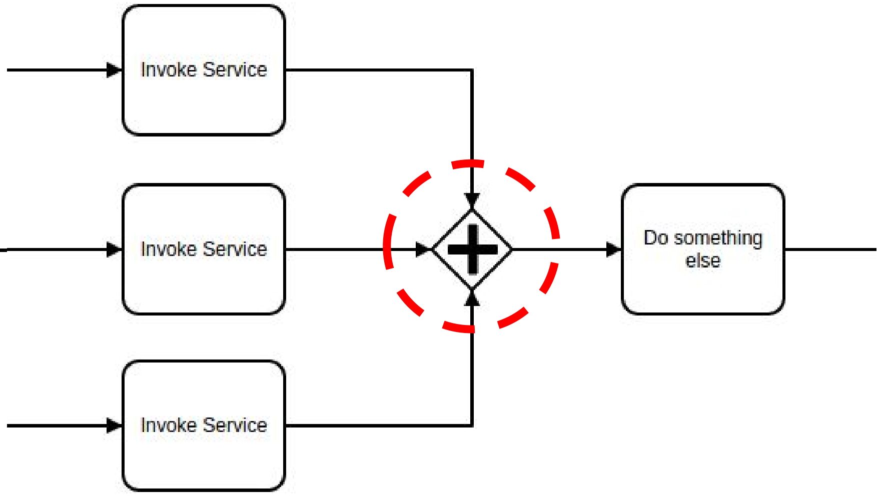
Mutual Exclusion



Same Mechanics for all concurrency “Conflicts”



“Racing incoming Signals”



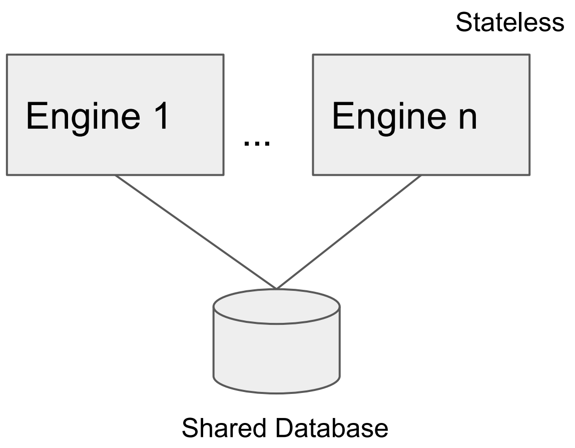
Synchronization

Good Reasons for this Design

Relational Database solves all the “hard problems”

Good integration into transactional + Thread oriented Programing Models

(Spring, Java EE, …)



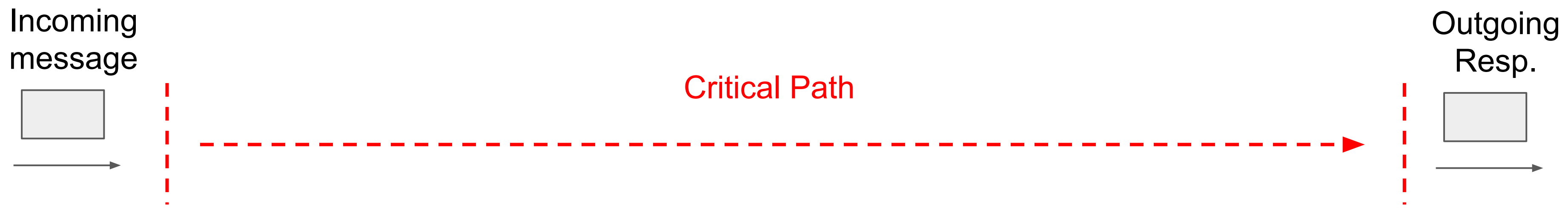
Very simple Clustering Model

Downsides

- Cannot really move away from RDBMS
- Not “hardware friendly”
- Inadequate outside of transactional programming model (OLEs with non-transactional side effects)

=> Slow and limited in scalability

How can we move past this?


Incoming message
Critical Path
Outgoing Resp.

Take all “secondary features” out of the critical path

- History / Audit log
- Human Tasks
- Service Invocations
- Sync I/O
- ...

=> Do these asynchronously

Employ the single writer principle

- There can be only one thread which is able to change the state of a Workflow Instance
- Use intelligent correlation to route messages to this thread

=> No concurrency conflicts
=> No locks either (neither optimistic nor pessimistic)

Use Batch I/O around the critical path to amortize on expensive I/O costs.

- First level of persistence can be a filesystem journal
- Fault tolerance
- Second level (cold store) is an actual database. Written to asynchronously, out of the critical path, actual waitstates only.

Compose the system out of simple, single purpose agents which communicate using bounded queues / ring buffers

- simple code = fast code
- cache friendly
- use modern CPUs and Memory Subsystems efficiently

Current Status

Working on prototypes.

Start discussions with potential early adopters.

Talking here today ... :)

We need your Input!

Comment on my Blog: **http://long-running.net**

Online survey:
https://goo.gl/sShBvy

goto; conference

Please

Remember to rate session

Thank you!