EVENT SOURCING WITH JVM LANGUAGES

Rahul Somasunderam

JavaOne 2017

ABOUT ME

ABOUT ME

- Wanted Event Sourcing in 2016

ABOUT ME

- Wanted Event Sourcing in 2016
- Didn’t like anything I saw

ABOUT ME

- Wanted Event Sourcing in 2016
- Didn’t like anything I saw
- Built my own thing - Grooves

ABOUT ME

- Wanted Event Sourcing in 2016
- Didn’t like anything I saw
- Built my own thing - Grooves
- I solve problems in Health Care

WHERE I WORK

Transcend Insights® empowers population health management with The Platform Above—technology that rises above the data silos and connects care teams with the insights they need to succeed in a value-based care world.

ABOUT YOU

ABOUT YOU

ABOUT YOU

ABOUT YOU

ABOUT YOU

WHAT IS EVENT SOURCING

Treat your database like you treat your application logs

WHAT DOES THAT MEAN

Create

Read

~~Update~~

~~Delete~~

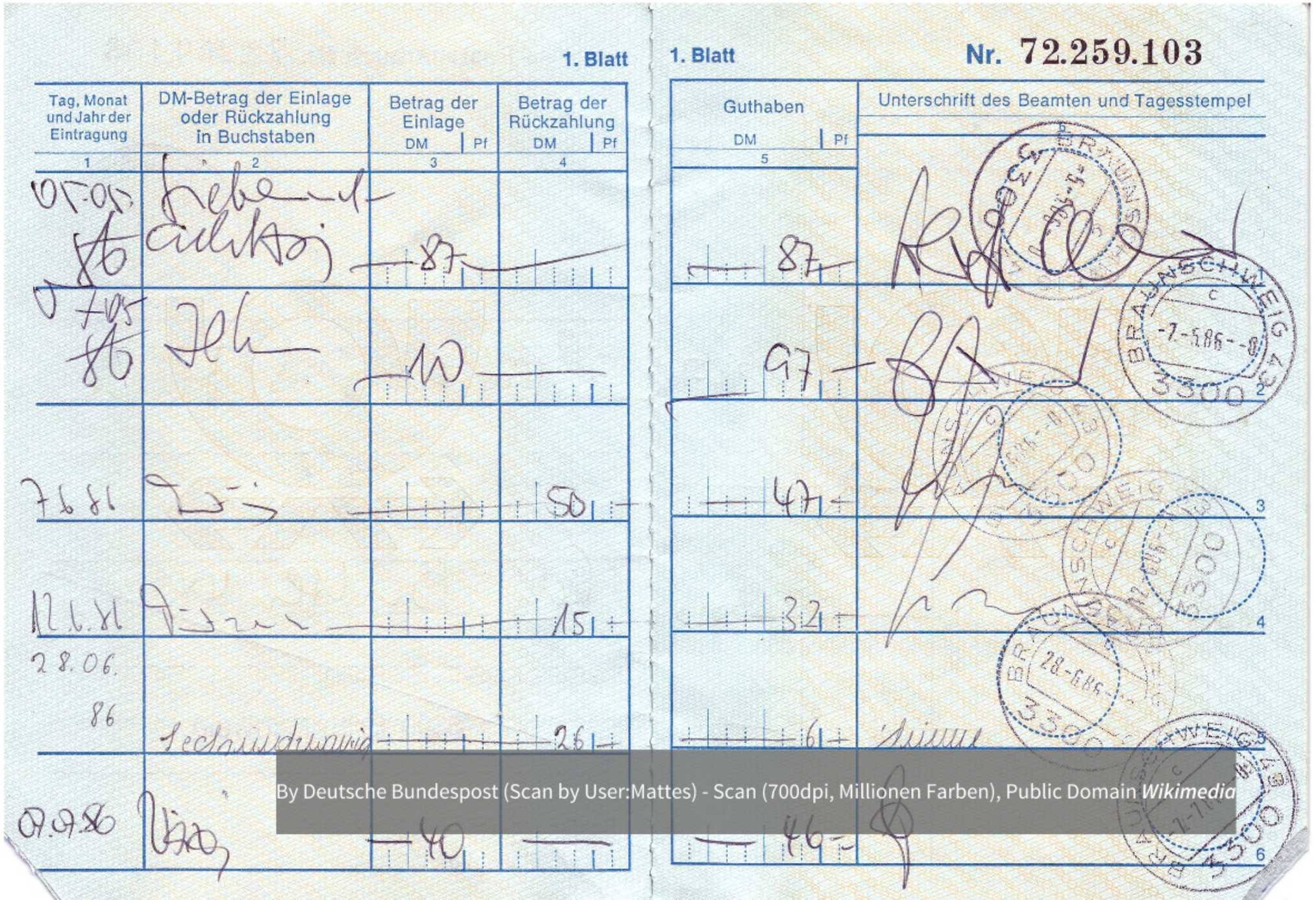
BAD EXAMPLE 1

Mark and Jane share a bank account. The account has a balance of $75. Mark goes to an ATM and withdraws $50. At the same time Jane attempts to withdraw $50.

This is what our code looks like

```
1   @Transactional
    boolean withdraw(String accountNumber, double amount) {
      double balance = accountService.getBalance(accountNumber);
      if (balance > amount) {
        balance -= account;
        accountService.setBalance(accountNumber, balance);
        return true;
      } else {
        return false;
      }
    }
```

WHAT BANKS TYPICALLY DO



BUT THAT WAS BANKING

BUT THAT WAS BANKING

If you can look at your logs and debug your application, you are already doing that.

BUT THAT WAS BANKING

If you can look at your logs and debug your application, you are already doing that.

Banks defined their business model this way hundreds of years ago.

WHAT ABOUT MY DOMAIN?

```
/* Aggregate */
class Account {
  String accountNumber;
}
```

```
/* Events */
abstract class Transaction {
  Account account;
}
class AtmWithdrawal
    extends Transaction {
  String location;
  double amount;
}
class AtmDeposit
    extends Transaction {
  String location;
  double amount;
}
```

```
/* Snapshot */
class AccountSummary {
  double balance;
}
```

WHAT ABOUT MY DOMAIN?

```
/* Aggregate */
class Patient {
  String identifier;
  String system;
}
```

```
/* Events */
abstract class PatientEvent {
  Patient patient;
}
class MedicationPrescribed
    extends PatientEvent {
  String code;
  int quantity;
}
class ProcedurePerformed
    extends PatientEvent {
  String code;
}
```

```
/* Snapshot */
class PatientSummary {
  List<String> medications
  List<String> procedures;
}
```

COMPUTING SNAPSHOTS

$$S_N = f(S_0, E_1 .. E_N)$$

S_N Snapshot at Version N

S_0 empty snapshot

$E_1 .. E_N$ events from position 1 through N

f query

BUT IT’S NOT THAT SIMPLE

INCREMENTAL COMPUTATION

This is what we would like to do

$$S_N = f(S_K, E_{K+1} .. E_N)$$

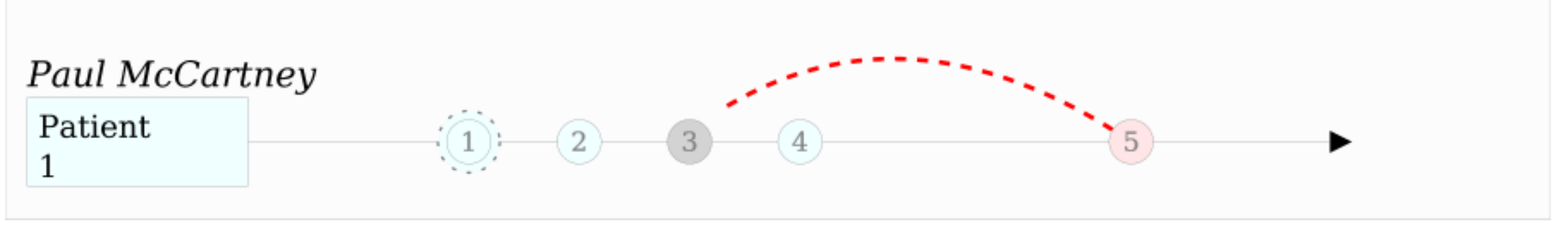
In Mathematics, it's called the distributive property

$$a + b + c = (a + b) + c$$

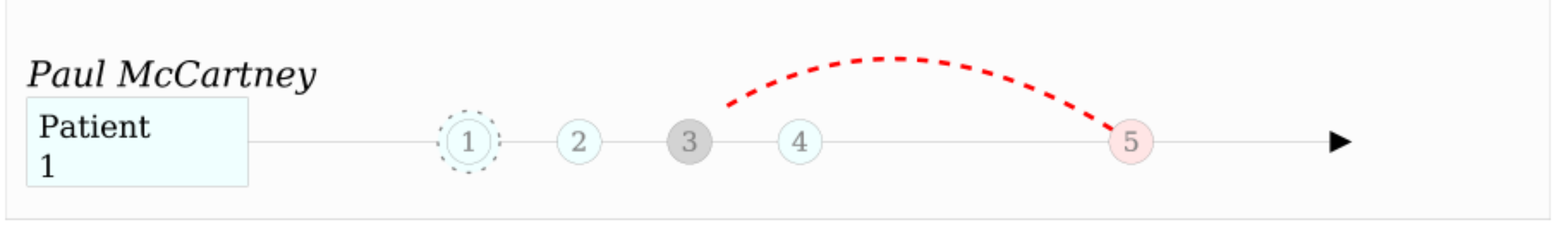
WE’RE NOT DONE YET

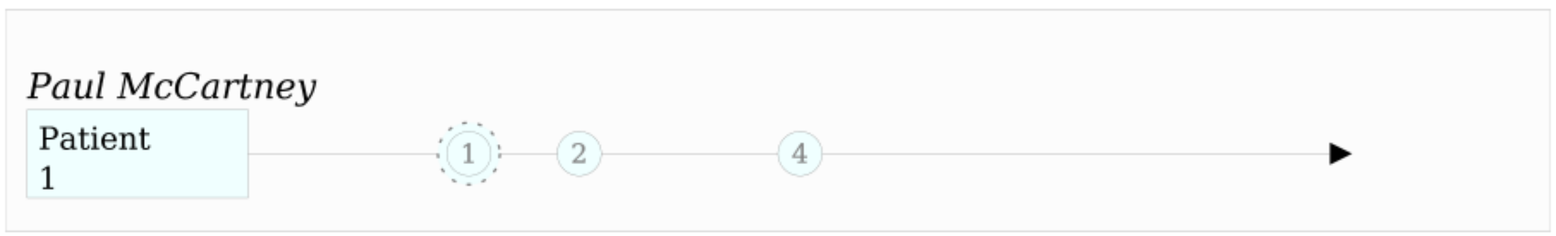
There are special events

REVERT

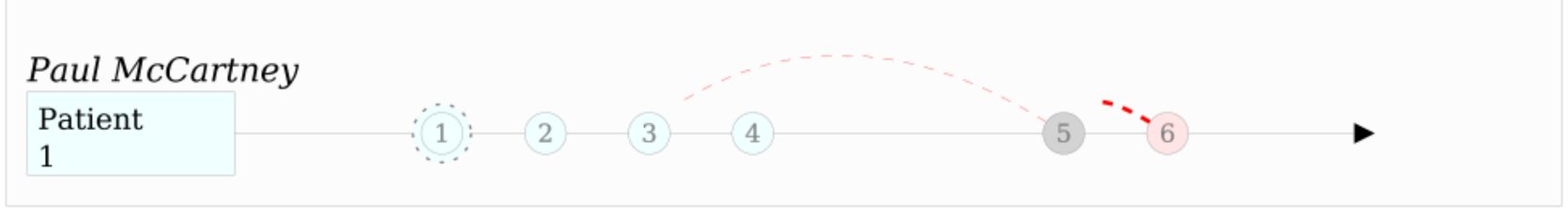


REVERT

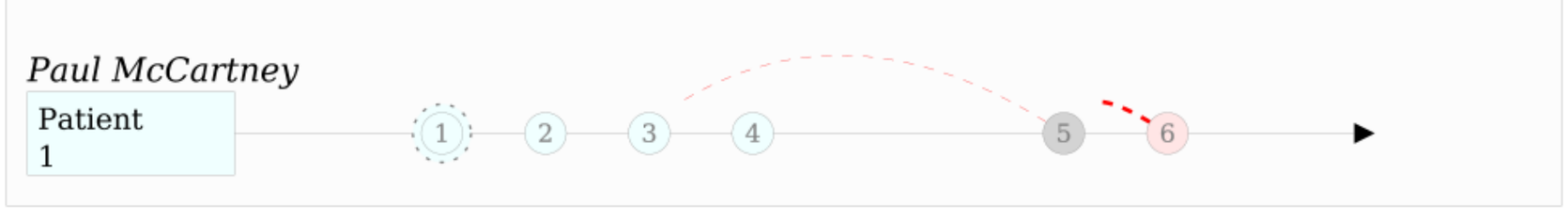

Paul McCartney
Patient 1
1
2
3
4
5

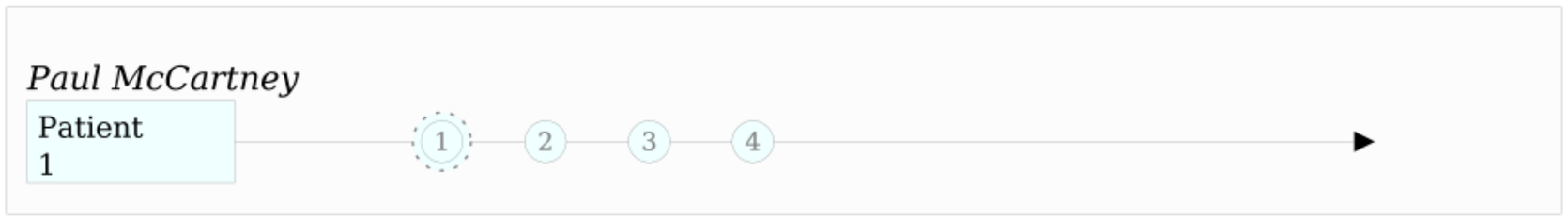

Paul McCartney
Patient 1
1
2
4

REVERT AGAIN

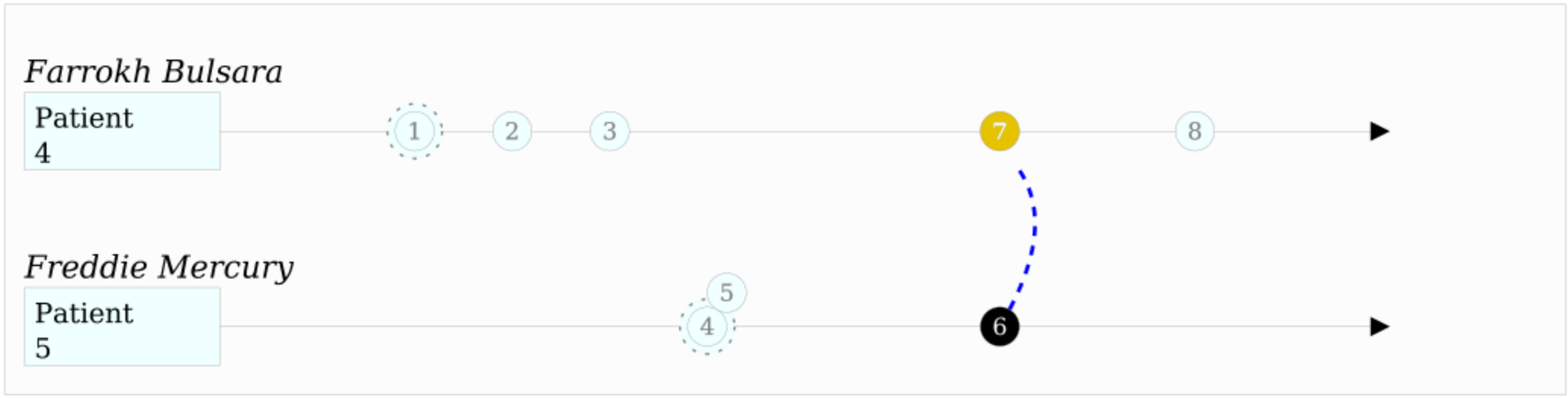


REVERT AGAIN

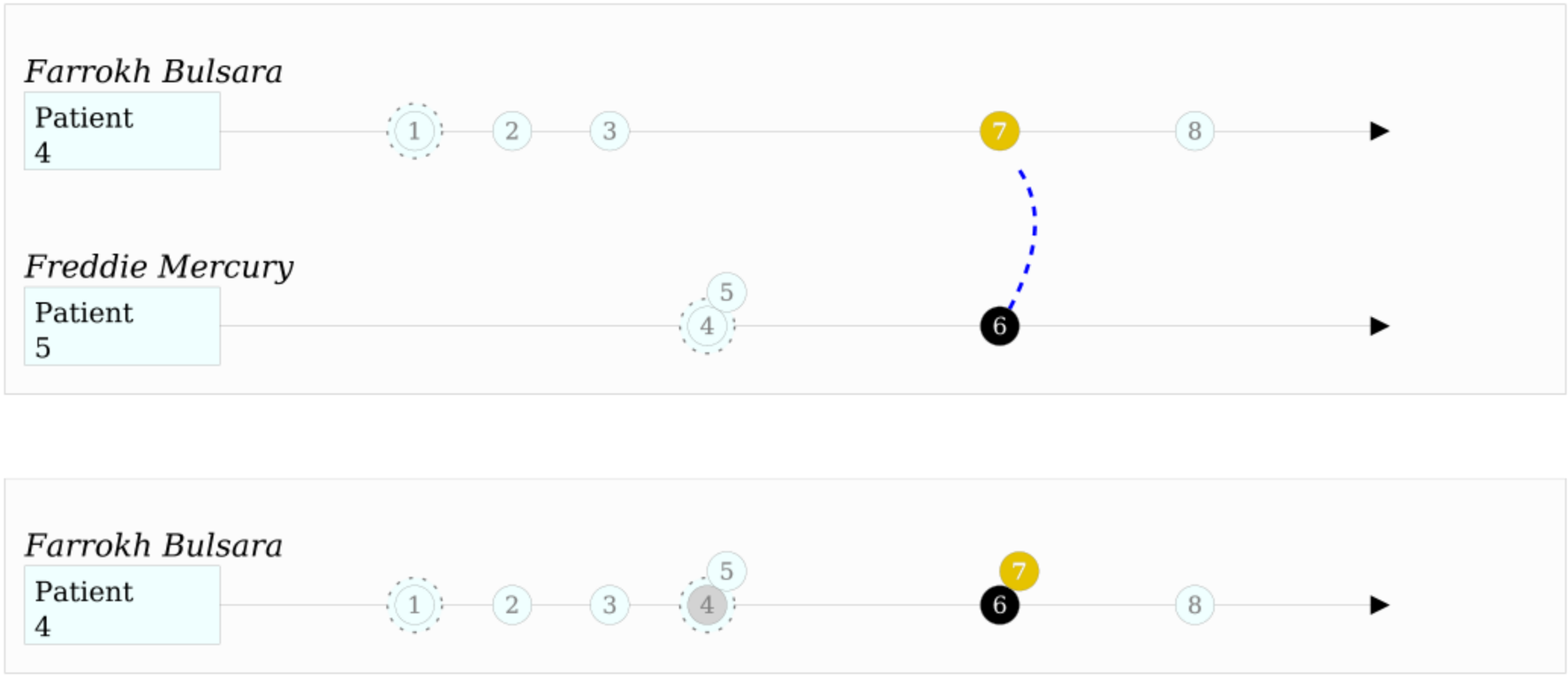




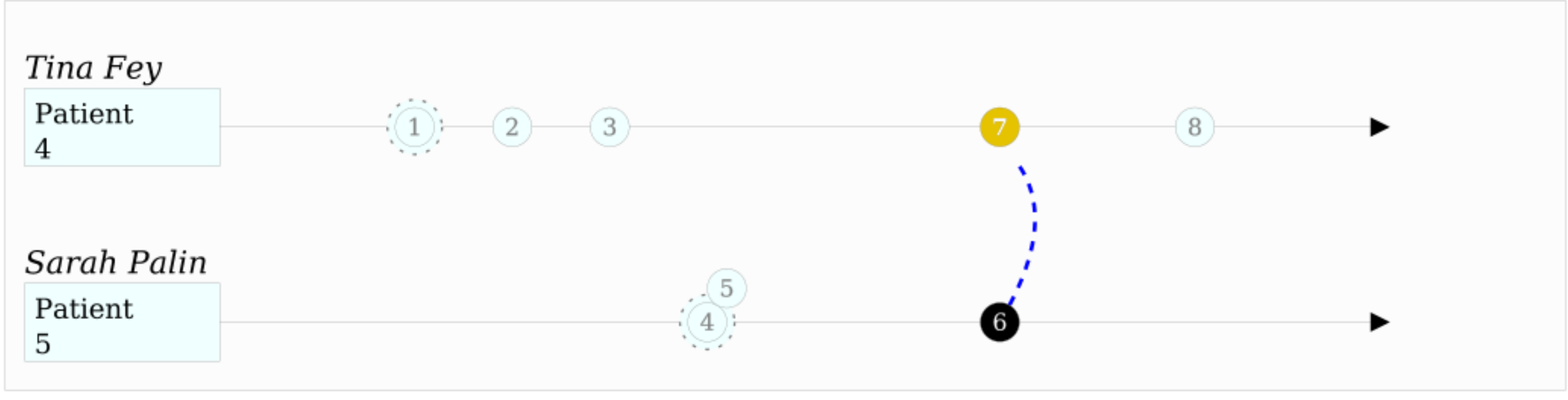
MERGE/DEPRECATE



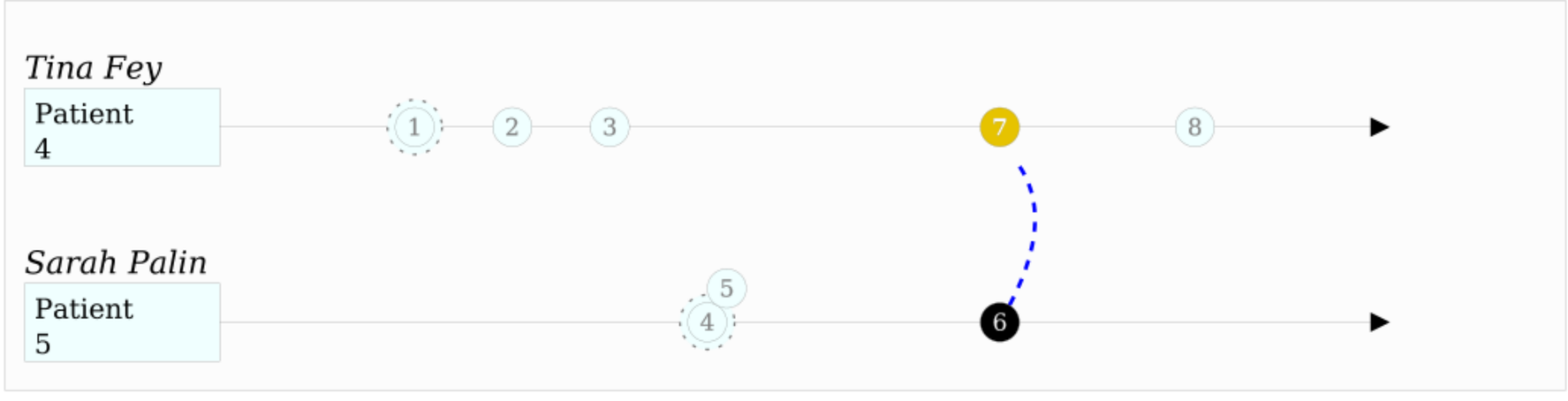
MERGE/DEPRECATE

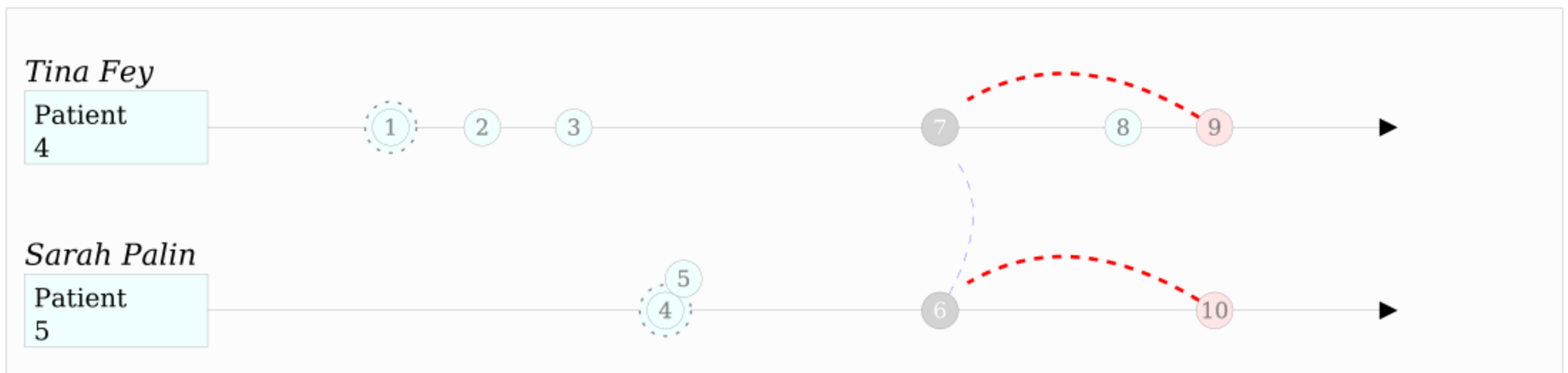


REVERTING A MERGE



REVERTING A MERGE





BAD EXAMPLE 2

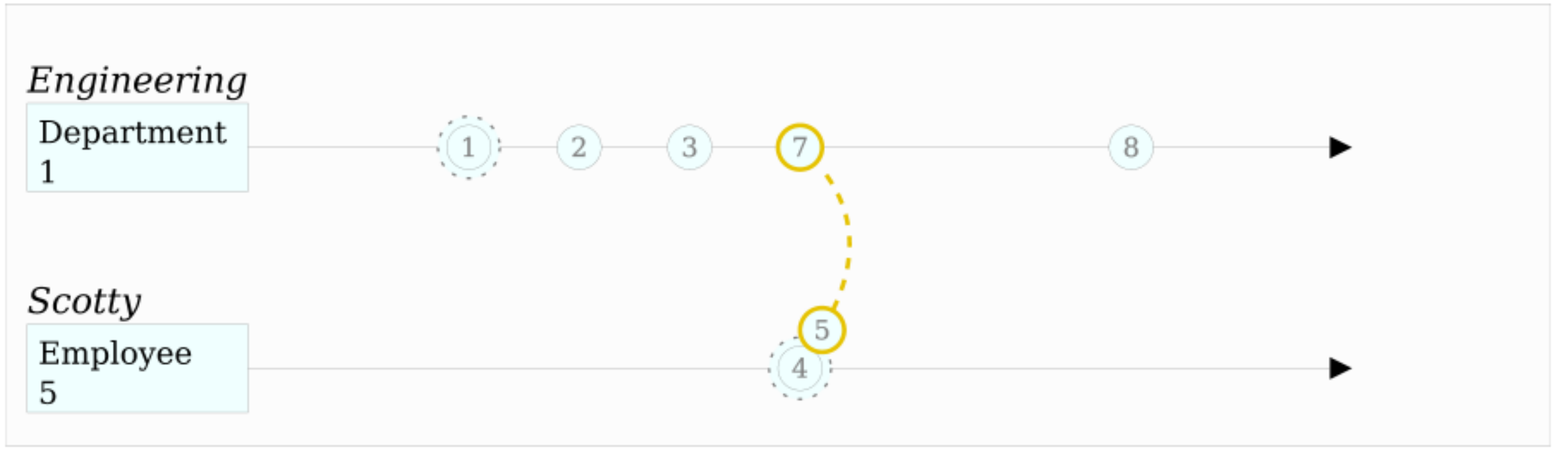
Employee

ID	EMP_NAME	DEPT_ID
1	Mr Spock	1
2	Scotty	2
3	Kirk	3
4	Janeway	3
5	La Forge	2

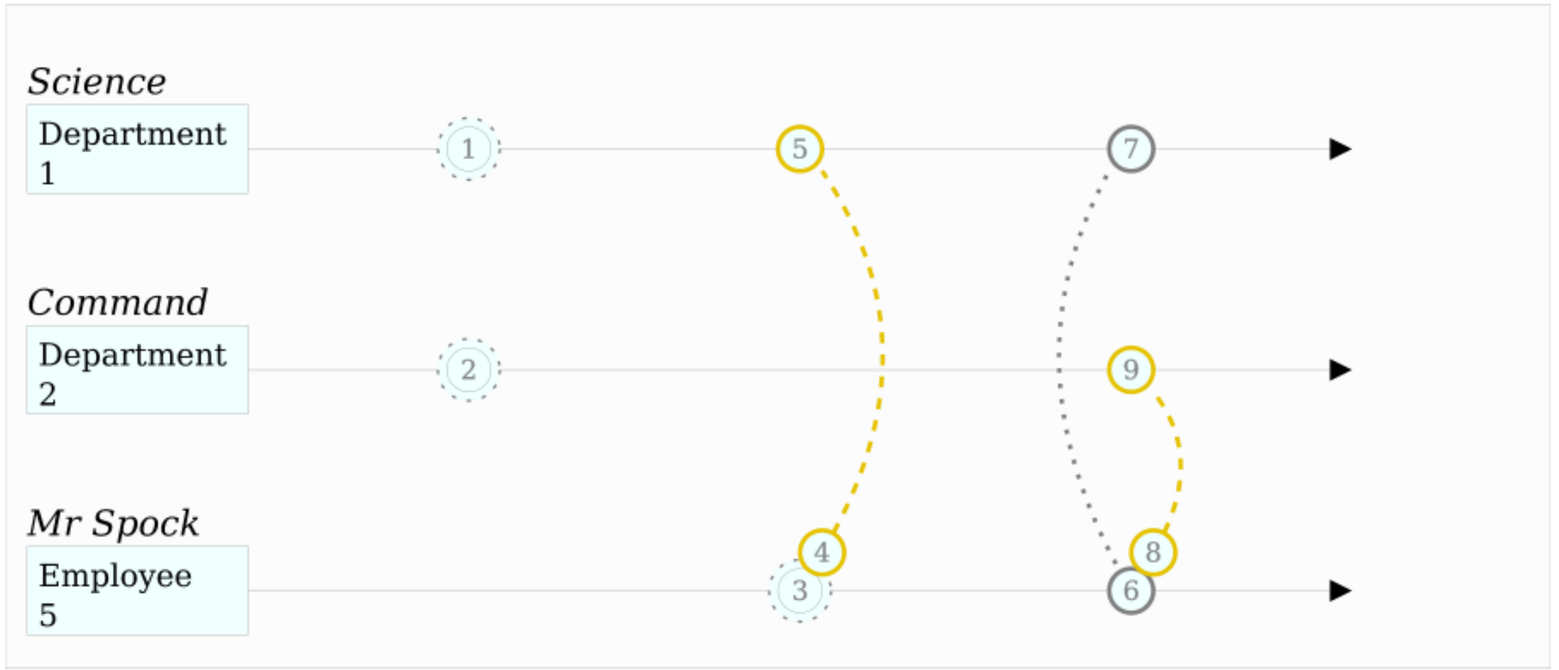
Department

ID	DEPT_NAME
1	Science
2	Engineering
3	Command

JOINS IN EVENT SOURCING



DISJOINS TOO



GROOVES DOMAIN OBJECTS

AGGREGATES

```
public class Patient implements AggregateType<Long> {
    private Long id;
    private String uniqueId;
}
```

EVENTS

```
public abstract class PatientEvent implements
        BaseEvent<Long, Patient, Long, PatientEvent> {
    private Patient aggregate;
    private Long id;
    private String createdBy;
    private RevertEvent<Long, Patient, Long, PatientEvent> revertedBy;
    private Date timestamp;
    private Long position;

    @Override
    @NotNull
    public Observable<Patient> getAggregateObservable() {
        return aggregate != null ? just(aggregate) : empty();
    }
}
```

REAL EVENTS

```
public class PatientCreated extends PatientEvent {
    private String name;
}
```

SPECIAL EVENTS

```
public class PatientEventReverted
        extends PatientEvent
        implements RevertEvent<Long, Patient, Long, PatientEvent> {
    private Long revertedEventId;
}
```

SNAPSHOTS

```
public class PatientAccount
        implements JavaSnapshot<Long, Patient, Long, Long, PatientEvent>,
        Serializable {
    private Long id;
    private Patient aggregate;
    private Patient deprecatedBy;
    private List<Patient> deprecates = new ArrayList<>();
    private Long lastEventPosition;
    private Date lastEventTimestamp;

    private String name;
    private BigDecimal balance = new BigDecimal(0);
    private BigDecimal moneyMade = new BigDecimal(0);

    public Observable<Patient> getAggregateObservable() {
        return just(aggregate);
    }
```

GROOVES QUERIES

SATISFYING GROOVES

```
public class PatientAccountQuery<...> extends QuerySupport<...> {
    ...
}
```

Could also be `VersionedQuerySupport` or `TemporalQuerySupport`

FETCHING SNAPSHOTS

```
@Override
public PatientAccount createEmptySnapshot() {
    return new PatientAccount();
}

@Override
public Observable<PatientAccount> getSnapshot(
        long maxPosition, Patient aggregate) {
    ...
}

@Override
public Observable<PatientAccount> getSnapshot(
        Date maxTimestamp, Patient aggregate) {
    ...
}
```

FETCHING EVENTS

```
    @Override
    public Observable<PatientEvent> getUncomputedEvents(
            Patient aggregate, PatientAccount lastSnapshot, long version) {

    }
```

```
    @Override
    public Observable<PatientEvent> getUncomputedEvents(
            Patient aggregate, PatientAccount lastSnapshot, Date snapshotTim

    }
```

HANDLING ERRORS

```
    @Override
    default Observable<EventApplyOutcome> onException(
            Exception e, PatientAccount snapshot, PatientEvent event) {
        getLog().error("Error computing snapshot", e);
        return just(CONTINUE);
    }
```

PROCESSING EVENTS

For languages like Java, Groovy

```
public Observable<EventApplyOutcome> applyPatientCreated(
        PatientCreated event, PatientAccount snapshot) {
    if (snapshot.getAggregate() == event.getAggregate()) {
        snapshot.setName(event.getName());
    }
    return just(CONTINUE);
}

...
```

PROCESSING EVENTS

For languages with case classes

```
override fun applyEvent(
    event: PatientEvent.Applicable, snapshot: PatientAccount) =
        when (event) {
            is PatientEvent.Applicable.Created -> {
                // Your logic here
                just(CONTINUE)
            }
            is PatientEvent.Applicable.ProcedurePerformed -> {
                // Your logic here
                just(CONTINUE)
            }
            is PatientEvent.Applicable.PaymentMade -> {
                // Your logic here
                just(CONTINUE)
            }
        }
```

EVENTS

For languages with case classes

```
sealed class PatientEvent : BaseEvent<..> {
    // Properties and methods from equivalent java class

    sealed class Applicable : PatientEvent() {
        data class Created(val name: String) : Applicable()
        data class ProcedurePerformed(
                val code: String, val cost: Double) : Applicable()
        ..
    }

    data class Reverted(override val revertedEventId: String) :
            PatientEvent(), RevertEvent<..>
}
```

COMPLETENESS FOR JAVA

...and Groovy.

```
@Aggregate public class Patient {...}
public abstract class PatientEvent {}

@Event(Patient.class)
public class ProcedurePerformed extends PatientEvent {}
@Event(Patient.class)
public class PaymentMade extends PatientEvent {}

@Query(aggregate=Patient.class, snapshot=PatientAccount.class)
public class PatientAccountQuery {
    ...
    Observable<EventApplyOutcome> applyProcedurePerformed() {...}
    Observable<EventApplyOutcome> applyPaymentMade() {...}
}
```

GOTCHAS

How many Tour de France General Classification Tour victories did this guy have on 2007-10-01?

EPA/JASPER JUINEN, 22 July 2004. Sourced from ***vosizneias***

LANCE ARMSTRONG

1992	Joins Motorola	2002	
1993	DNF	2003	
1994	DNF	2004	
1995	36	2005	
1996	DNF	2005	Retires
1996-10	Diagnosed with Cancer	2009	Returns from retirement
1997-02	Declared cancer free	2009	3
1998	Joins US Postal	2010	23
1999		2010	Retires
2000		2012-10	Stripped of all wins from 1998 through 2010
2001			

1. Based on what we knew in 2007, How many TdF GC Tour victories did he have on 2007-10-01?

2. Based on what we know in 2017, How many TdF GC Tour victories did he have on 2007-10-01?


1. Based on what we knew in 2007, How many TdF GC Tour victories did he have on 2007-10-01?
2. Based on what we know in 2017, How many TdF GC Tour victories did he have on 2007-10-01?
Ask the right question
UNITED STATES POSTAL SERVICE
BERRY FLOOR
TREK
AMD

ADVICE ON HANDLING DATA

- Be very careful about what your events look like.
- Don’t worry much about what your snapshots look like.

COMMUNICATION IS KEY

Scott Bellware
@sbellware

Follow

95% of the cost of event sourcing projects is explaining event sourcing

2:06 AM - Jun 11, 2017

10 142 294

NON EVENT SOURCES OF DATA

When the intent of the user is not clear from the datasource.

The Story of Goldilocks and The Three Bears, Award Publications LTD

THANK YOU

Slidedeck bit.ly/2017-es

Grooves github.com/rahulsom/grooves

Sun 15:00	**CON7610**	Microservices Data Patterns: CQRS and ES
Mon 16:30	**CON2526**	Reactive Stream Processing with Swarm and Kafka
Mon 17:30	**CON7474**	ES, Distributed Systems, and CQRS with Java EE
Tue 13:15	**CON4083**	Async by Default, Synchronous When Necessary
Wed 14:45	**CON4277**	Three µS Patterns to Tear Down Your Monoliths