Real Time Recommendations using Spark Streaming

Elliot Chow

NETFLIX

Why?

- React more quickly to changes in interest
- Time-of-day effects
- Real-world events

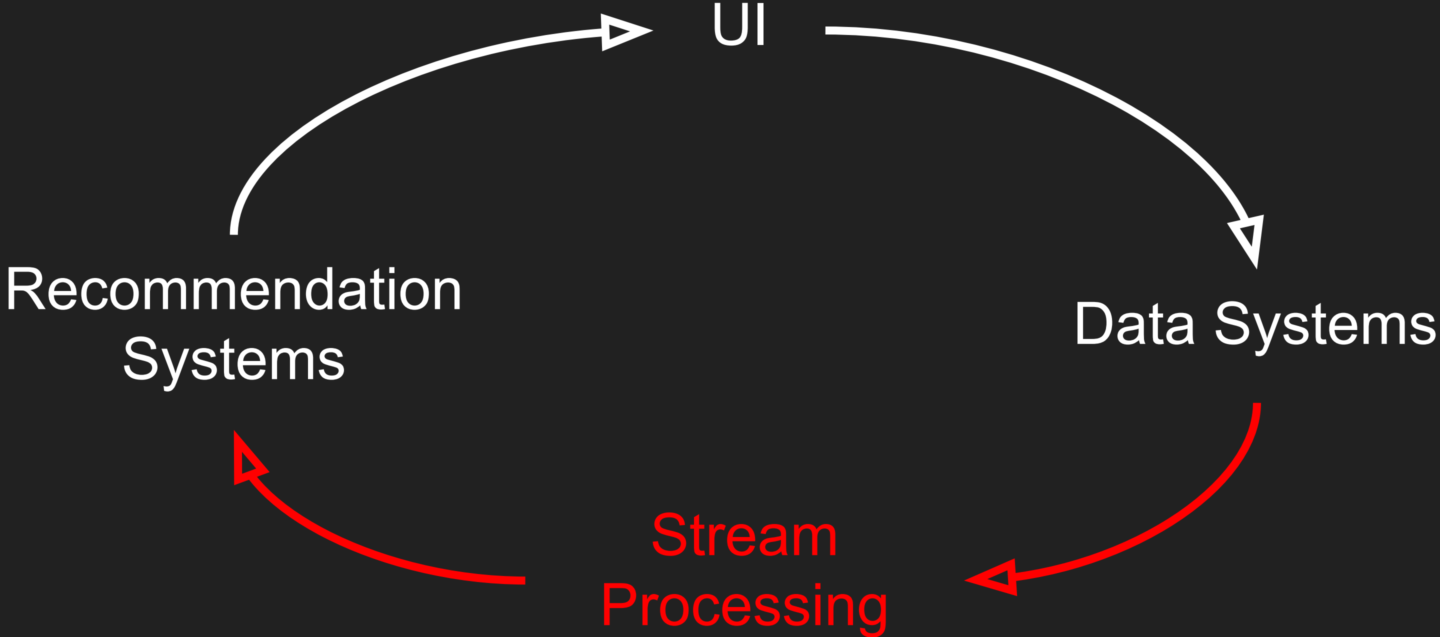
Trending Now

New Releases

Because you watched The Get Down

NETFLIX

Feedback Loop



NETFLIX

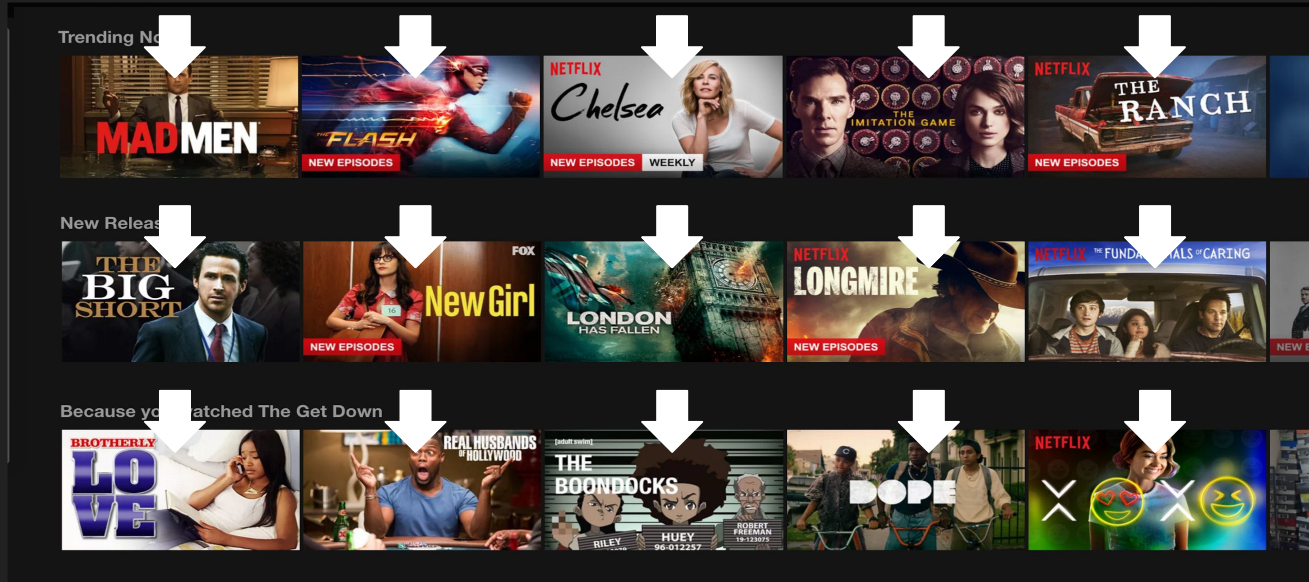
Trends Data

- What people browse: impressions
- What people watch: plays

NETFLIX

Trends Data - Impressions

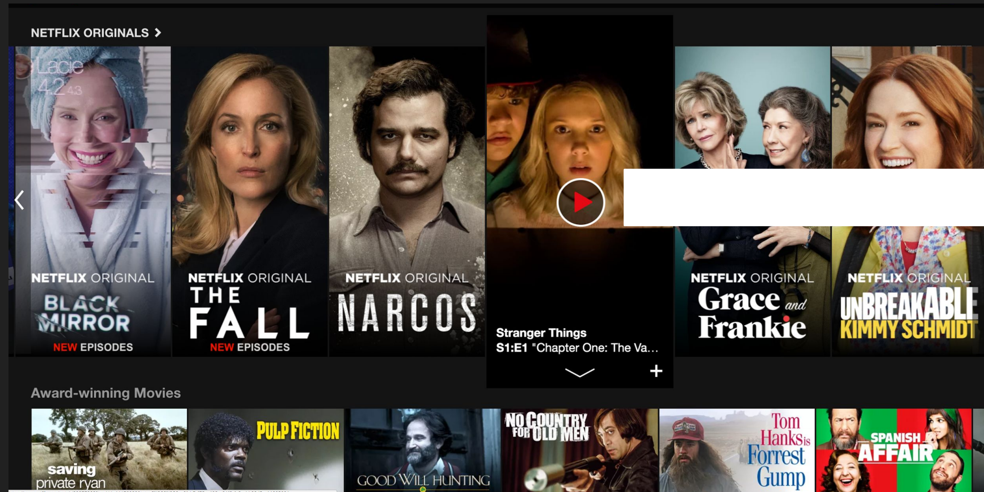
Appearance of a video in the viewport



NETFLIX

Trends Data - Plays

Member plays a video



NETFLIX

Why Spark Streaming?

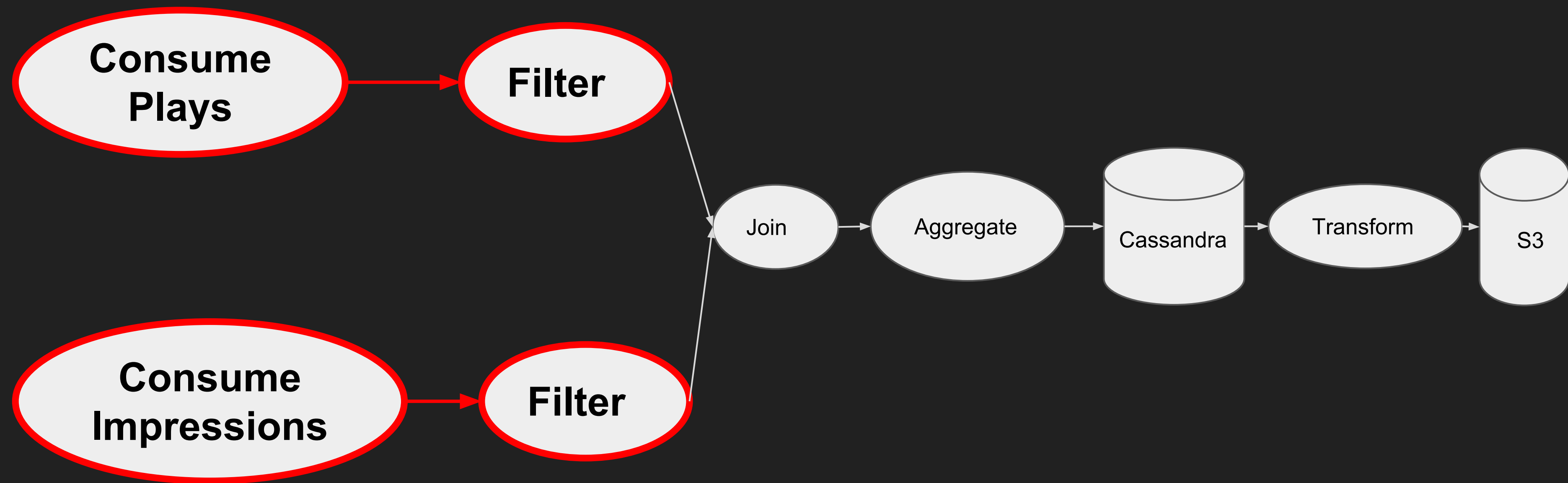
- Existing Spark infrastructure
- Experience with Spark
- Batch and Streaming

NETFLIX

Components

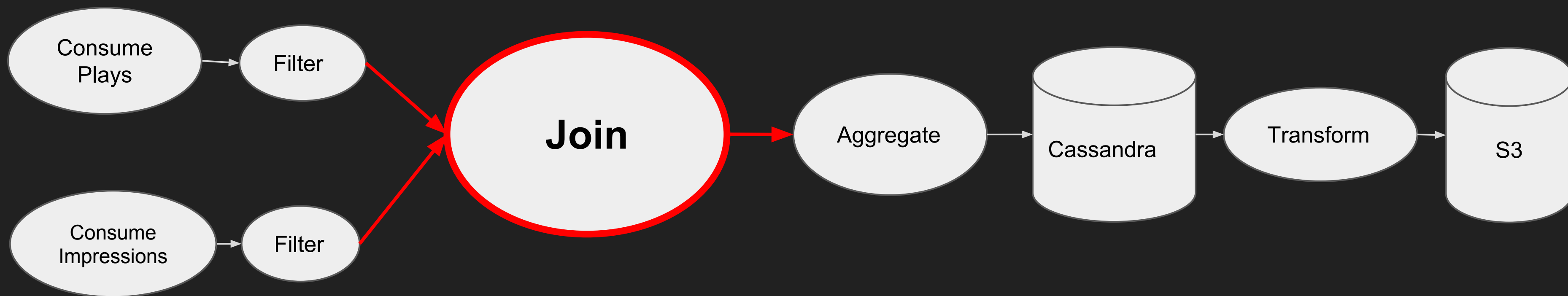
NETFLIX

Design



NETFLIX

Design



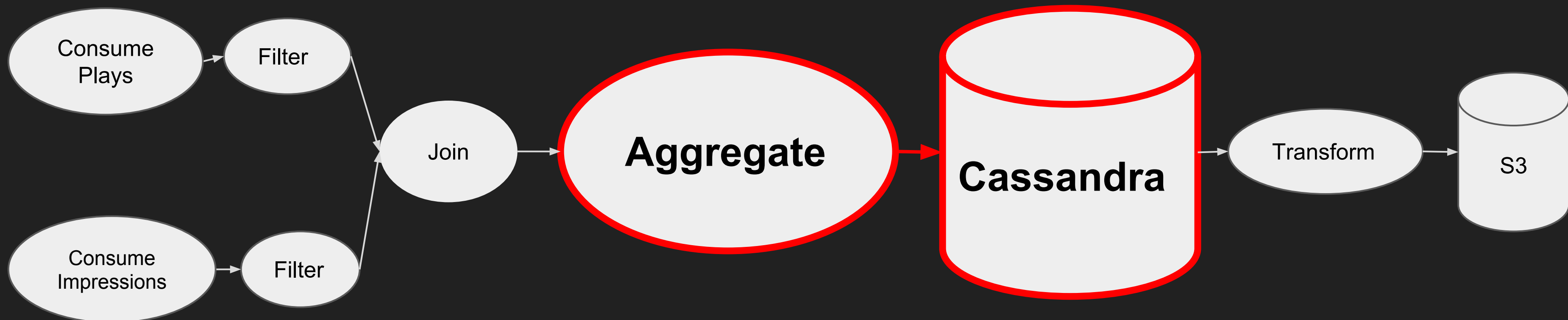
NETFLIX

Join Key

“Request Id” - a unique identifier of the source of a play or impression

NETFLIX

Design



NETFLIX

Output

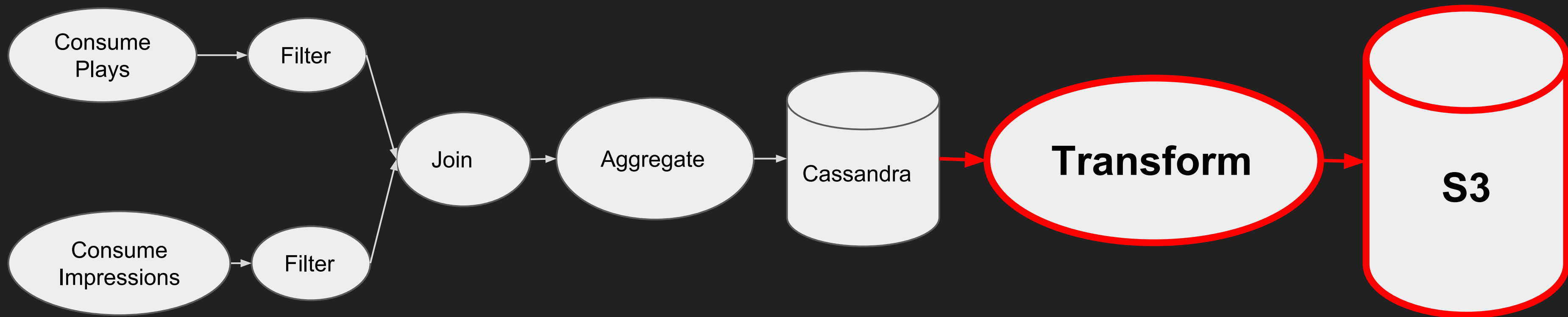
Video	Epoch	Plays	Impressions
Stranger Things	1 (00:00-00:30)	4	5
Stranger Things	1 (00:00-00:30)	3	6
House Of Cards	2 (00:30-01:00)	8	10
Marseille	2 (00:30-01:00)	3	3

NETFLIX

Output

- Instead of raw counts, output sets of request ids
    - Count = cardinality of the set of request ids
- Idempotent counting

Design



NETFLIX

Programming with Spark Streaming

NETFLIX

Streaming Joins

NETFLIX

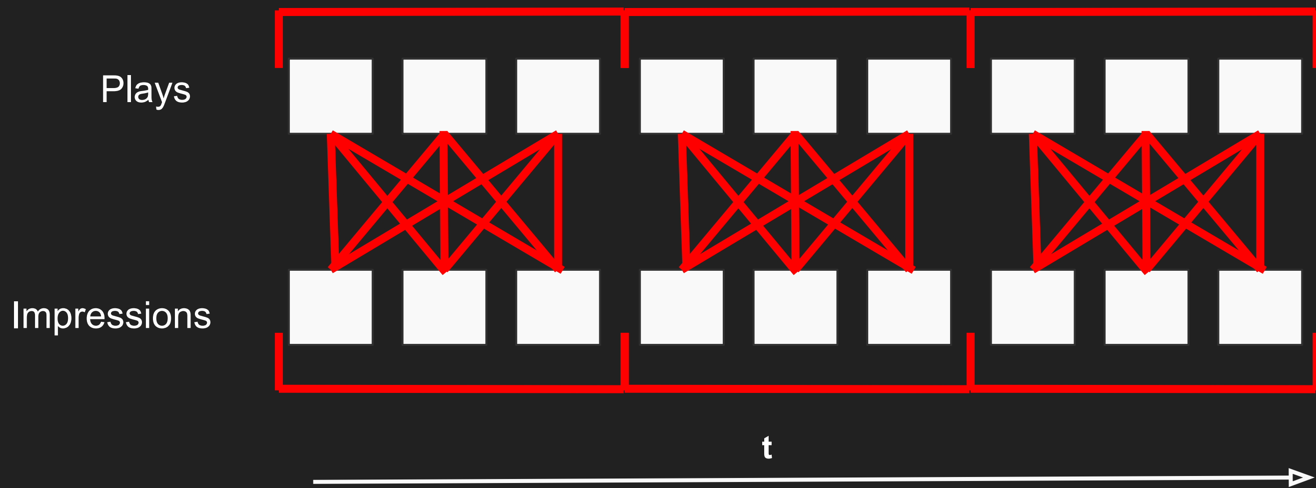
Streaming Joins - Time

- Time to browse and select a video
- Batched logging from client application
- Delays in data sources

NETFLIX

Streaming Joins - Attempt I

- Window both plays and impressions by epoch duration
- Join the two windows together
- Slide by epoch duration



NETFLIX

Streaming Joins - Attempt I

- Easy to implement
- Tight coupling with processing time
- Does not mesh well with absolute time windows
- Failure can mean loss of all data for the entire window

NETFLIX

Streaming Joins - Attempt II

- Join using mapWithState
- Join key is the mapWithState key
- State is the plays and impressions sharing the same join key
- Use timeouts to expire unjoined data

NETFLIX

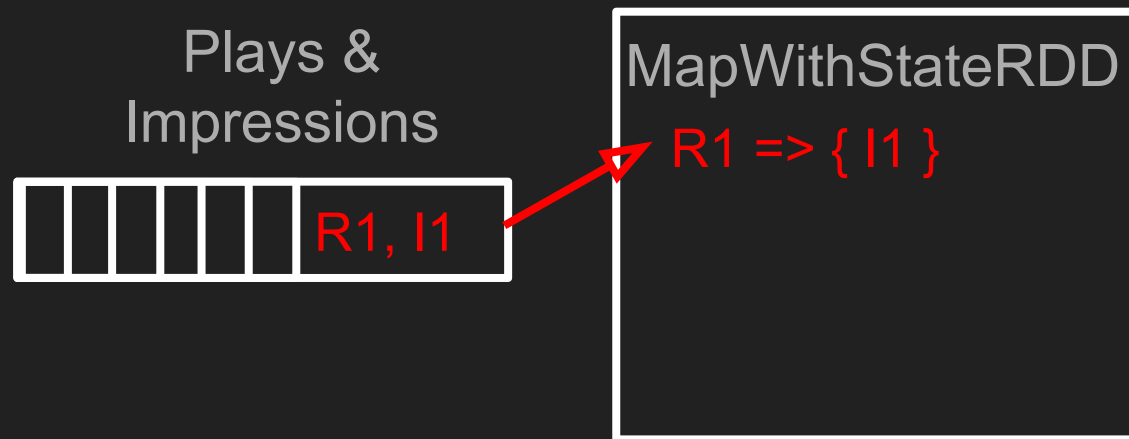
Streaming Joins - Attempt II

Plays &
Impressions

MapWithStateRDD

NETFLIX

Streaming Joins - Attempt II



NETFLIX

Streaming Joins - Attempt II

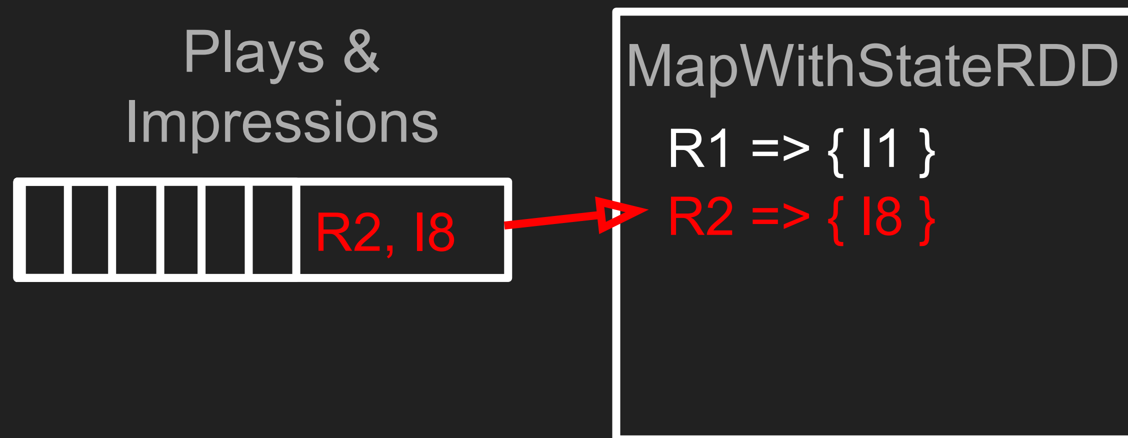
Plays &
Impressions

MapWithStateRDD

R1 => { I1 }

NETFLIX

Streaming Joins - Attempt II



NETFLIX

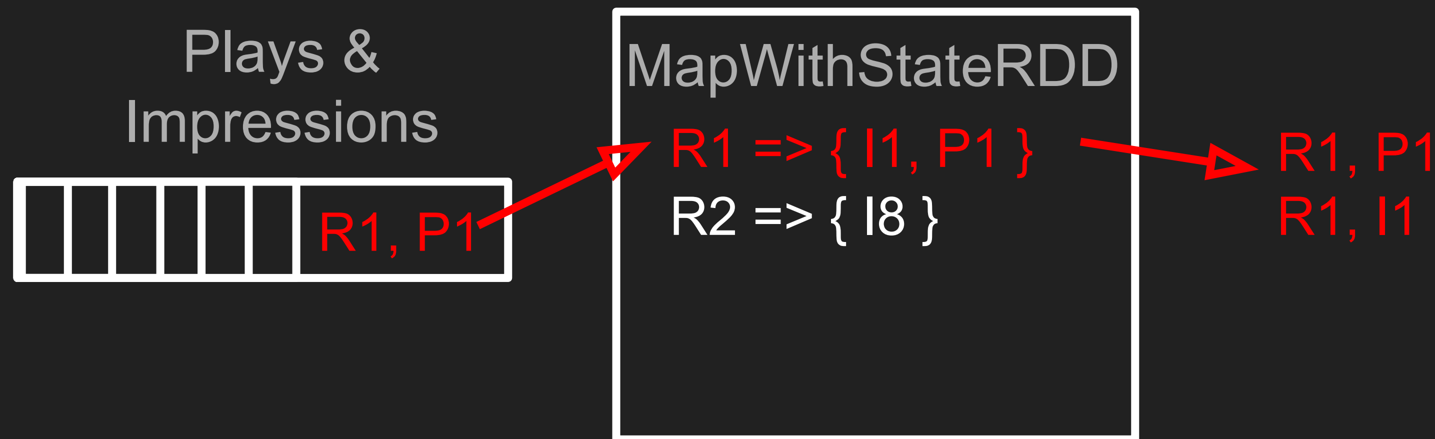
Streaming Joins - Attempt II

MapWithStateRDD

R1 => { I1 }
R2 => { I8 }

NETFLIX

Streaming Joins - Attempt II



NETFLIX

Streaming Joins - Attempt II

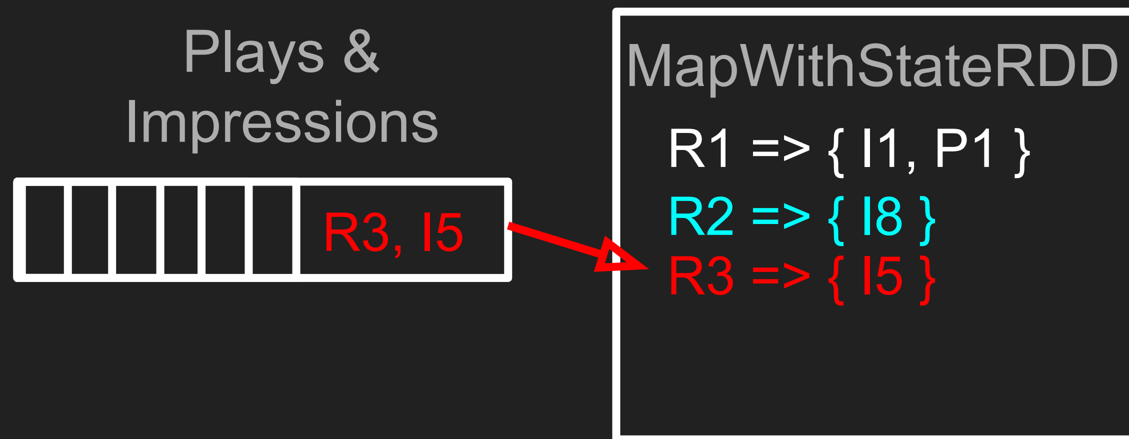
Plays &
Impressions

MapWithStateRDD

R1 => { I1, P1 }
R2 => { I8 }

NETFLIX

Streaming Joins - Attempt II



Streaming Joins - Attempt II

Plays &
Impressions

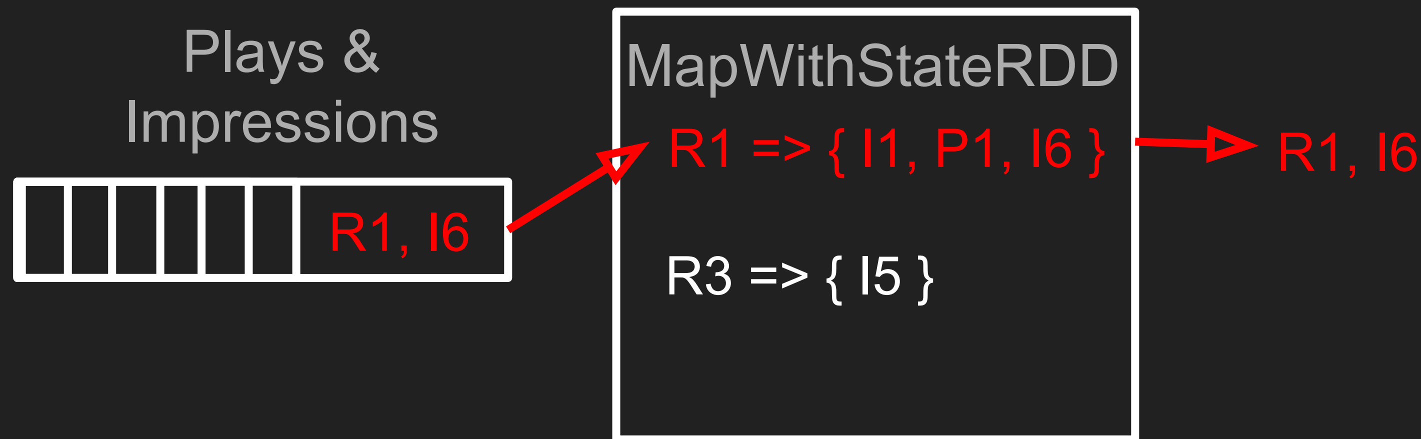
MapWithStateRDD

R1 => { I1, P1 }

R3 => { I5 }

NETFLIX

Streaming Joins - Attempt II



Streaming Joins - Attempt II

MapWithStateRDD

R1 => { I1, P1, I6 }

R3 => { I5 }

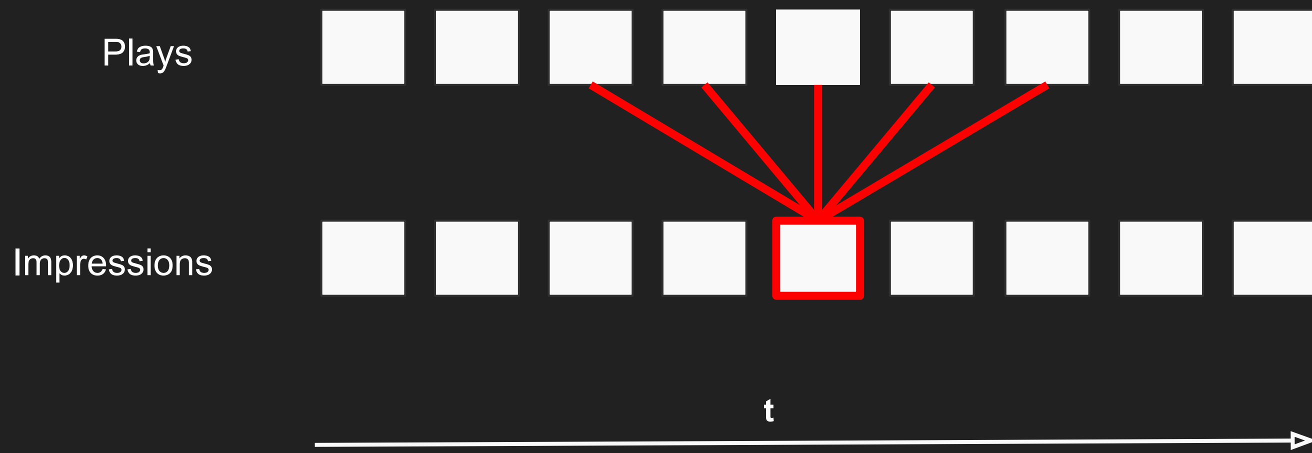
NETFLIX

Streaming Joins - Attempt II

- Make progress every batch
- Too much “uninteresting” data
- High memory usage
- Large checkpoints

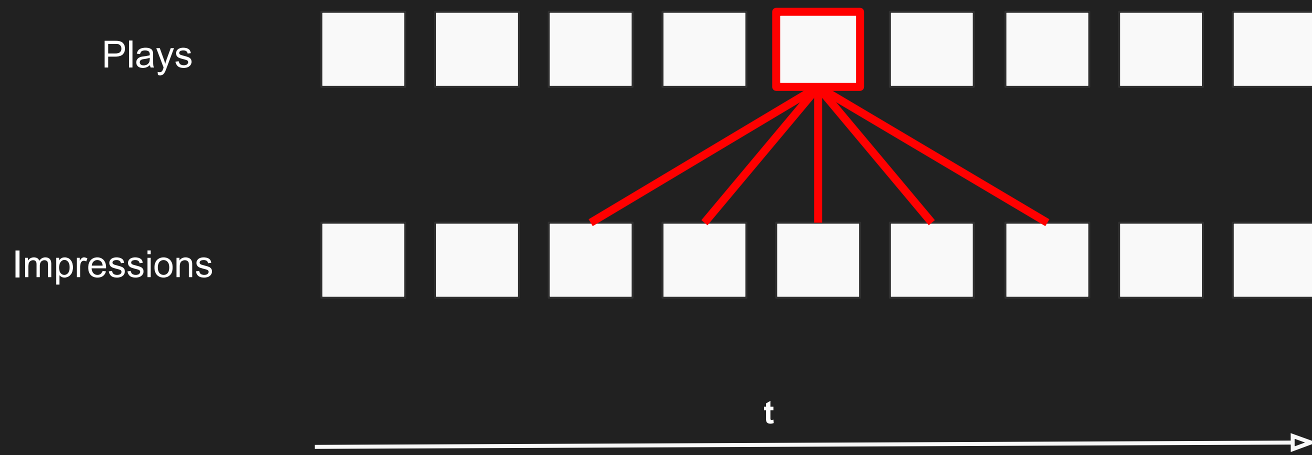
NETFLIX

Streaming Joins - An Observation



NETFLIX

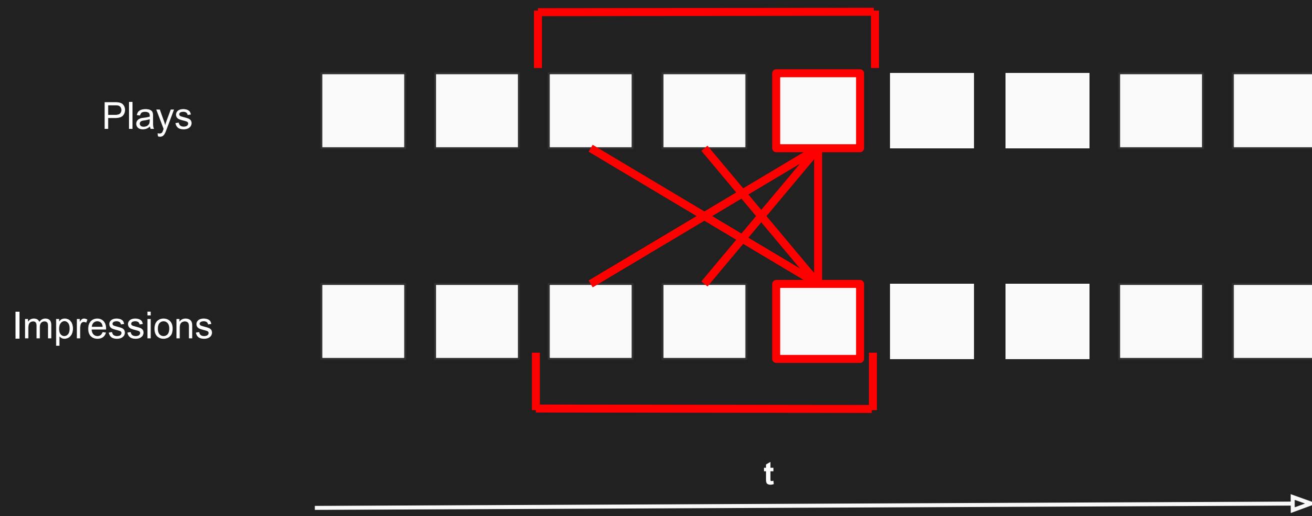
Streaming Joins - An Observation



NETFLIX

Streaming Joins - An Observation

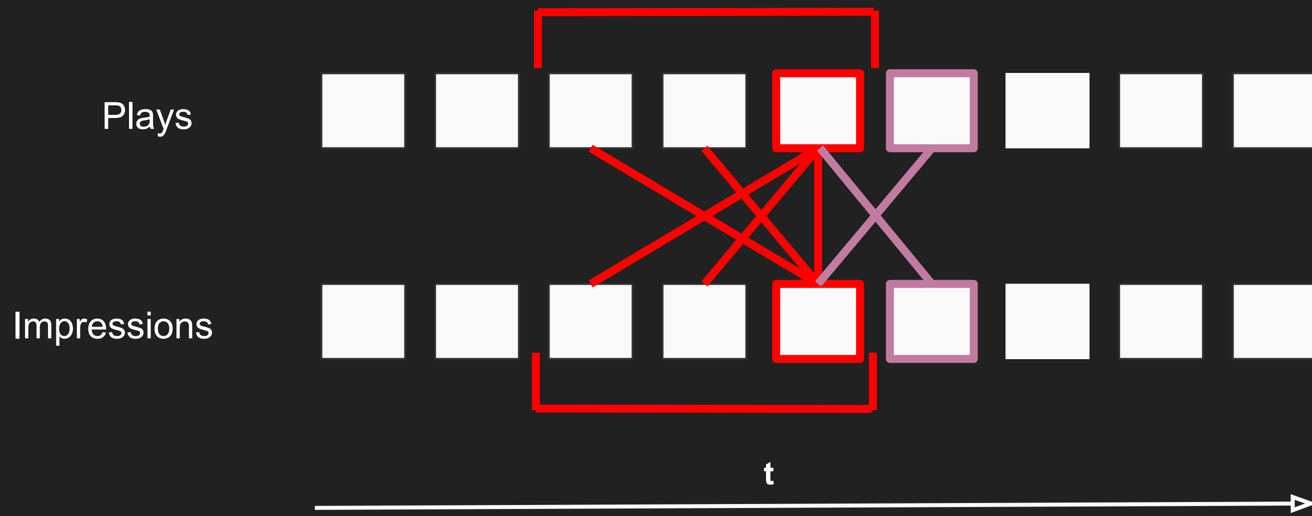
Join incoming batch of plays to windowed impressions, and vice versa



NETFLIX

Streaming Joins - An Observation

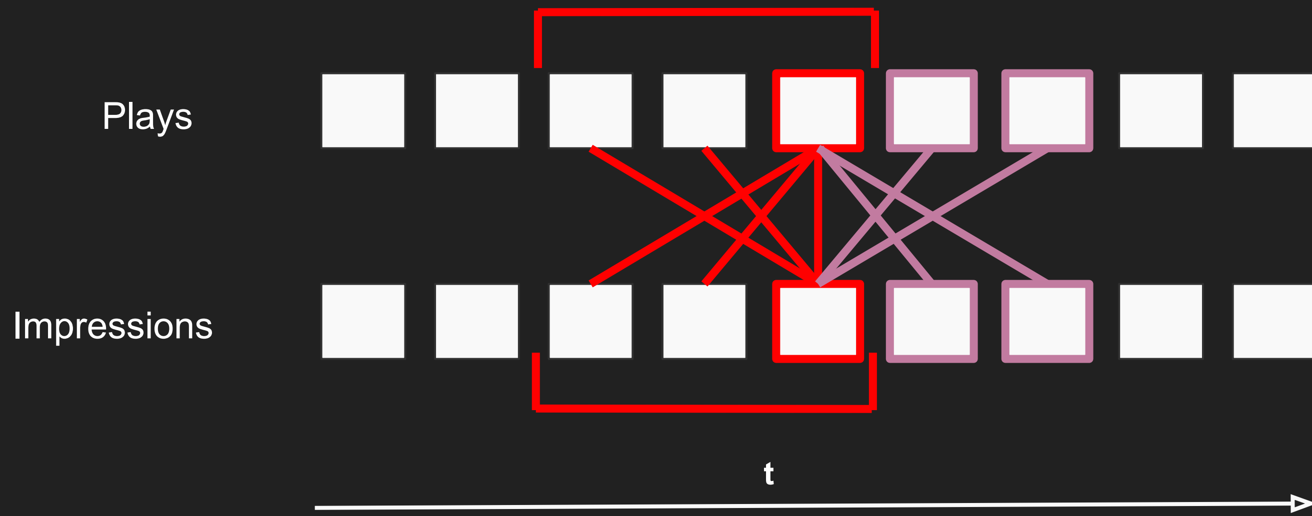
Slide by batch interval...



NETFLIX

Streaming Joins - An Observation

Slide by batch interval again...



NETFLIX

Streaming Joins - Attempt III

- Counts are updated every batch
- Uses Spark’s windowing
- No checkpoints

NETFLIX

mapWithState

mapWithState

- Can be used for more than sessionization

NETFLIX

mapWithState

- Can be used for more than sessionization
- Be aware of cache evictions
    - Lots of state may need to be recomputed

NETFLIX

mapWithState

```scala
val input: DStream[(VideoId, RequestId)] = // ...

val spec: StateSpec[VideoId, RequestId,
                    Set[RequestId],
                   (VideoId, Set[RequestId])] = // ...

val output: DStream[(VideoId, Set[RequestId])] = {
  input.
    mapWithState(spec)
}
```

NETFLIX

mapWithState

```scala
val input: DStream[(VideoId, RequestId)] = // ...

val spec: StateSpec[VideoId, RequestId,
                    Set[RequestId],
                   (VideoId, Set[RequestId])] = // ...

val output: DStream[(VideoId, Set[RequestId])] = {
 input.
    mapWithState(spec).
    groupByKey.
    mapValues(_.maxBy(_.size))
}
```

NETFLIX

mapWithState

```
val input: DStream[(VideoId, RequestId)] = // ...

val spec: StateSpec[VideoId, Iterable[RequestId],
                    Set[RequestId],
                    (VideoId, Set[RequestId])] = // ...

val output: DStream[(VideoId, Set[RequestId])] = {
  input.
    groupByKey.
    mapWithState(spec)
}
```

NETFLIX

mapWithState

```
val input: DStream[(VideoId, RequestId)] = // ...

val spec: StateSpec[VideoId, RequestId,
                    Set[RequestId], Unit] = // ...


val output: DStream[(VideoId, Set[RequestId])] = {
  input.
    mapWithState(spec).
    stateSnapshots
}
```

NETFLIX

Productionizing Spark Streaming

Metrics

- Monitoring system health
- Aid in diagnosis of issues
- Needs to be performant and accurate

NETFLIX

Metrics - Option I

- Use “traditional” stream processing metrics
    - Events/second, bytes/second, …
- Batching can make numbers hard to interpret
- Susceptible to recomputation

NETFLIX

Metrics - Option II

- Spark Accumulators
- Used internally by Spark
- Susceptible to recomputation
- Unclear when to report the metric
  - Can make use of SparkListener & StreamingListener

NETFLIX

Metrics - Option III

- Explicit counts on RDDs
- Counts will be accurate
- Additional latency
- Use caching to prevent duplicate work*

NETFLIX

Metrics

- Processing time < Batch interval
- Time the different parts of the job
    - Spark is lazy - may require forcing evaluation
- Use Spark UI metrics

NETFLIX

Error Handling

- What exceptions cause the streaming job to crash?

NETFLIX

Error Handling

- What exceptions cause the streaming job to crash?
  - Most seem to be caught to keep the job running
- Exception handling is application-specific
- Stop-gap: track the elapsed time since the batch started

NETFLIX

Future Work

Future Work

- Red/Black deployment with zero data-loss

NETFLIX

Future Work

- Red/Black deployment with zero data-loss
- Auto-scaling

NETFLIX

Future Work

- Red/Black deployment with zero data-loss
- Auto-scaling
- Improved back pressure per topic

NETFLIX

Future Work

- Red/Black deployment with zero data-loss
- Auto-scaling
- Improved back pressure per topic
- Updating broadcast variables

Questions?

We’re hiring!

elliot@netflix.com

NETFLIX