cppcon
the c++ conference

rti
Your systems. Working as one.

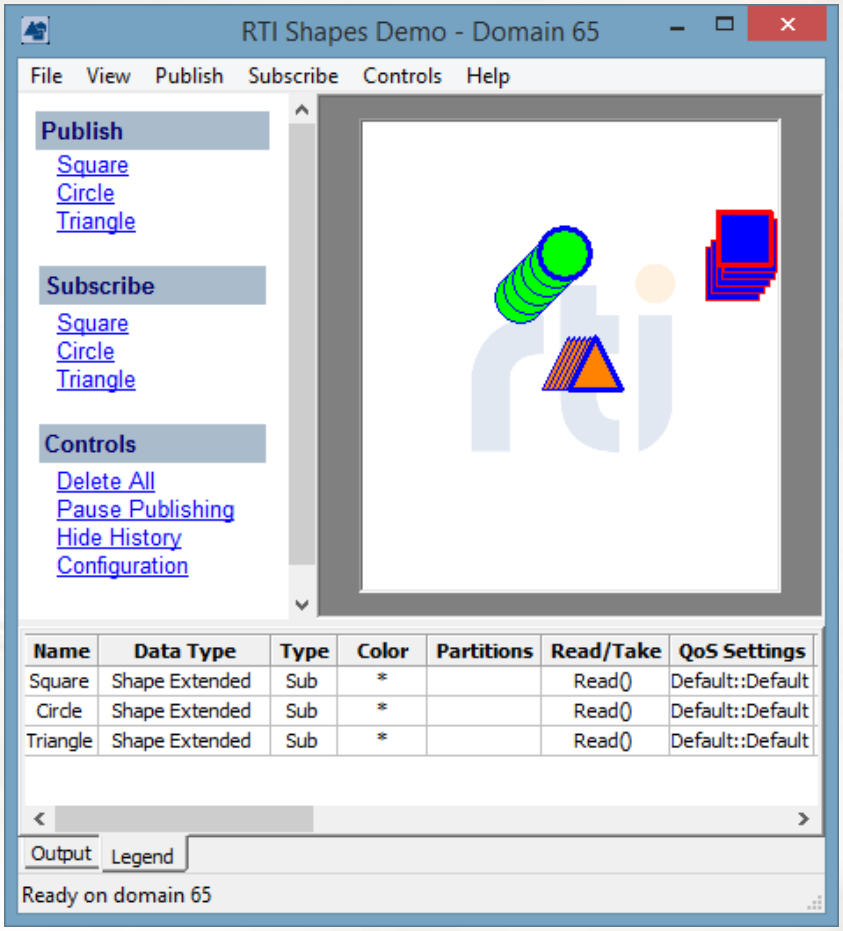
Reactive Stream Processing in Industrial IoT using DDS and Rx

Sumant Tambe, Ph.D.
Principal Research Engineer and Microsoft VC++ MVP
Real-Time Innovations, Inc.
@sutambe

Sept. 21, 2015

Outline

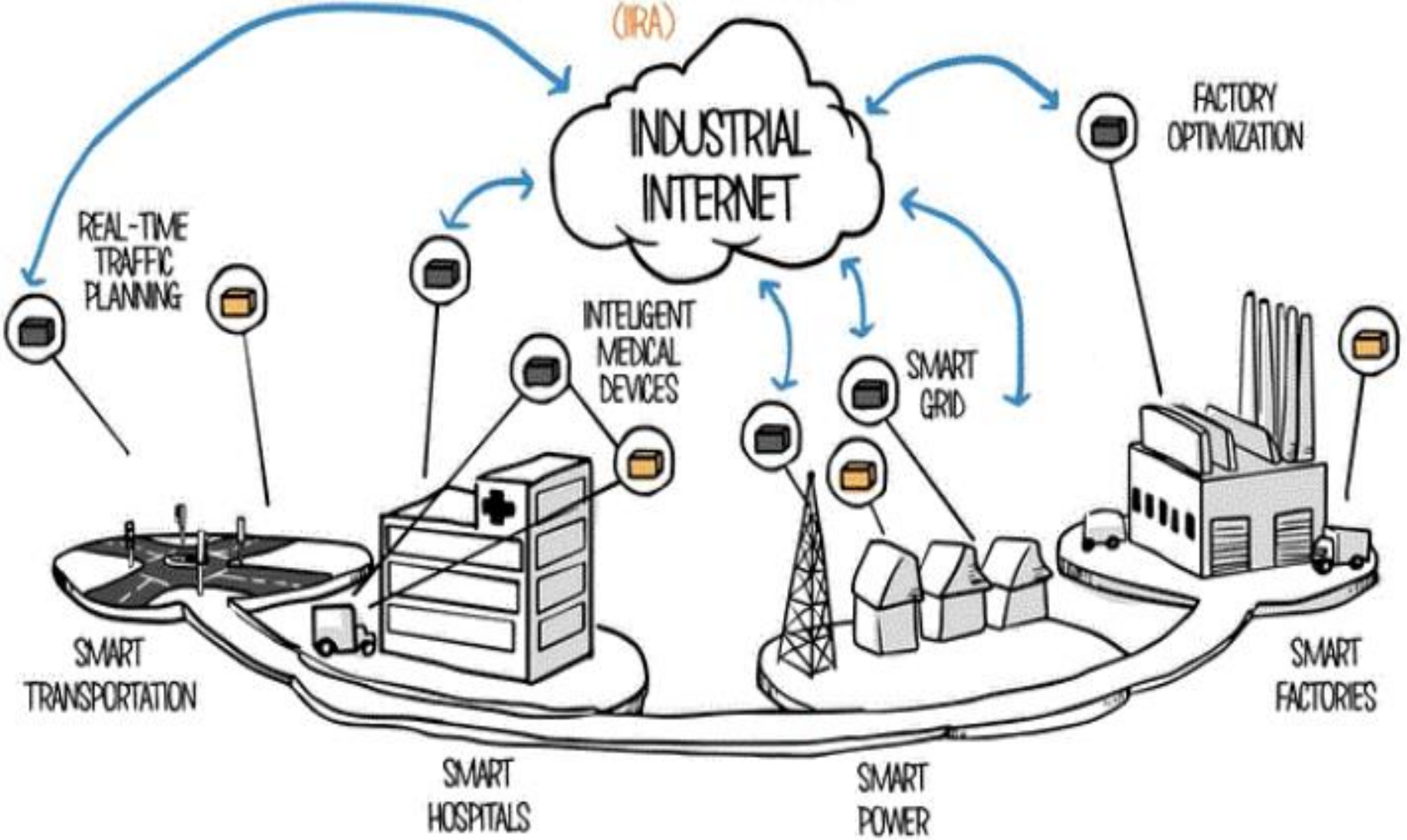
- Industrial IoT Systems
- Reactive Systems
- Data Distribution Service
- Stream Processing
- Reactive Extensions (Rx)
- Rx4DDS
- Stream Processing Demos with Shapes



Industrial Internet of Things

BLUEPRINT FOR THE INDUSTRIAL INTERNET

THE INDUSTRIAL INTERNET REFERENCE ARCHITECTURE (IIRA)



Industrial IoT Systems

Cannot stop processing the information.
Live within world-imposed timing.



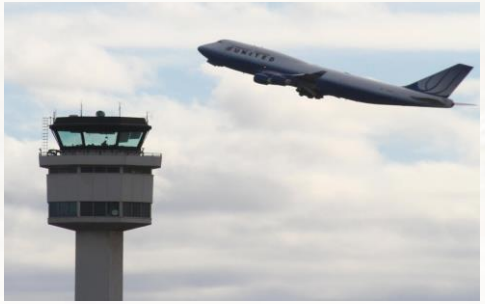





Image source: Google Images

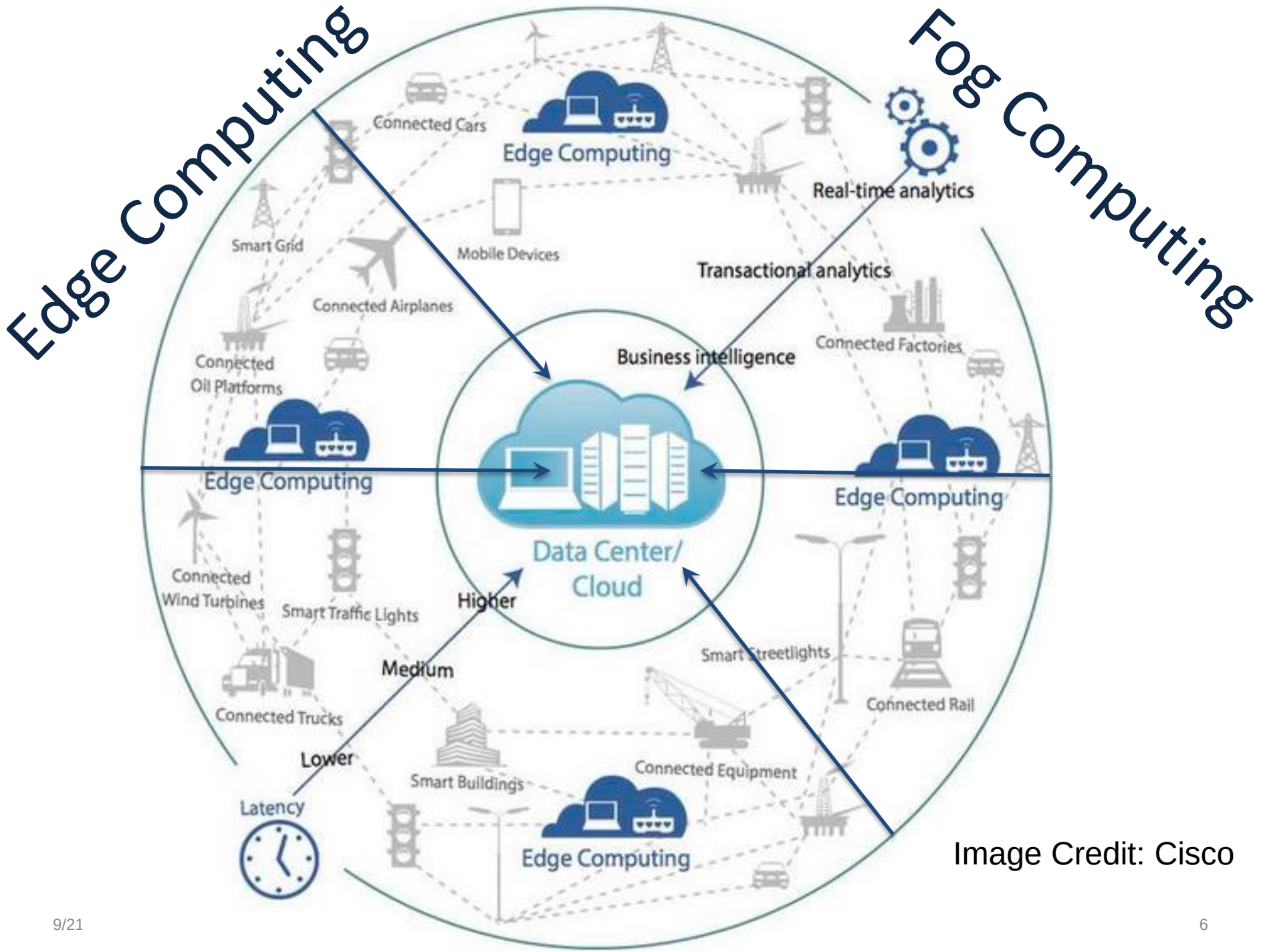


Image Credit: Cisco

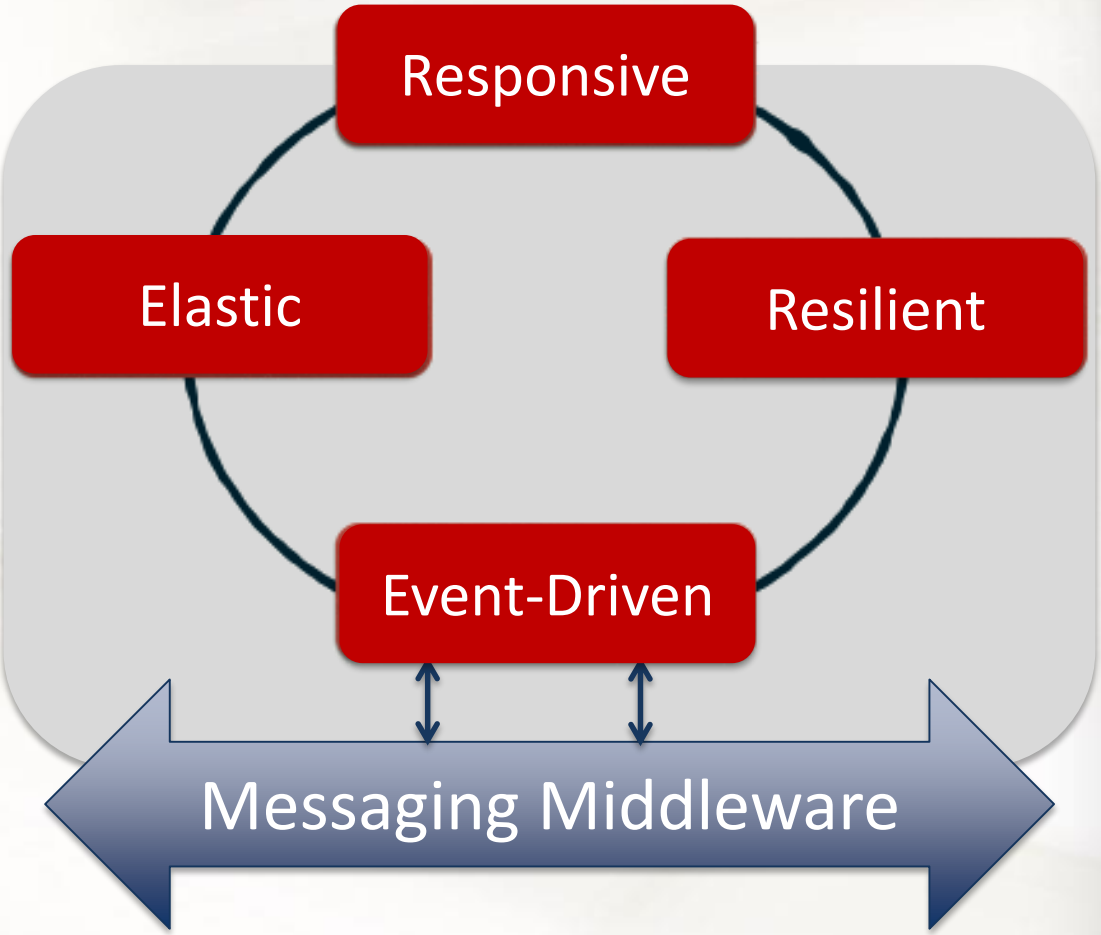
Reactive Systems

Responsive: Reacts to events at the speed of environment

Resilient: fault-tolerant

Elastic/Scalable: Scales easily up/down with load and cpu cores.

Event-Driven: asynchronous, loosely-coupled, modular, pipelined



Reactive Manifesto: www.reactivemanifesto.org

Reactive Manifesto

Jonas Bonér
@jboner

@gregyoung @mjpt777 Elasticity is decoupling in space, also known as distribution, while concurrency is decoupling in time.

8/16/15, 11:20 PM

Jonas Bonér
@jboner

@gregyoung @mjpt777 Concurrency not sufficient, you need a model that supports distribution onto multiple cores & machines. I.e. Elasticity.

8/16/15, 11:25 PM

Jonas Bonér
@jboner

@gregyoung @mjpt777 ...and location transparency can simplify the programming model for distribution (and resilience).

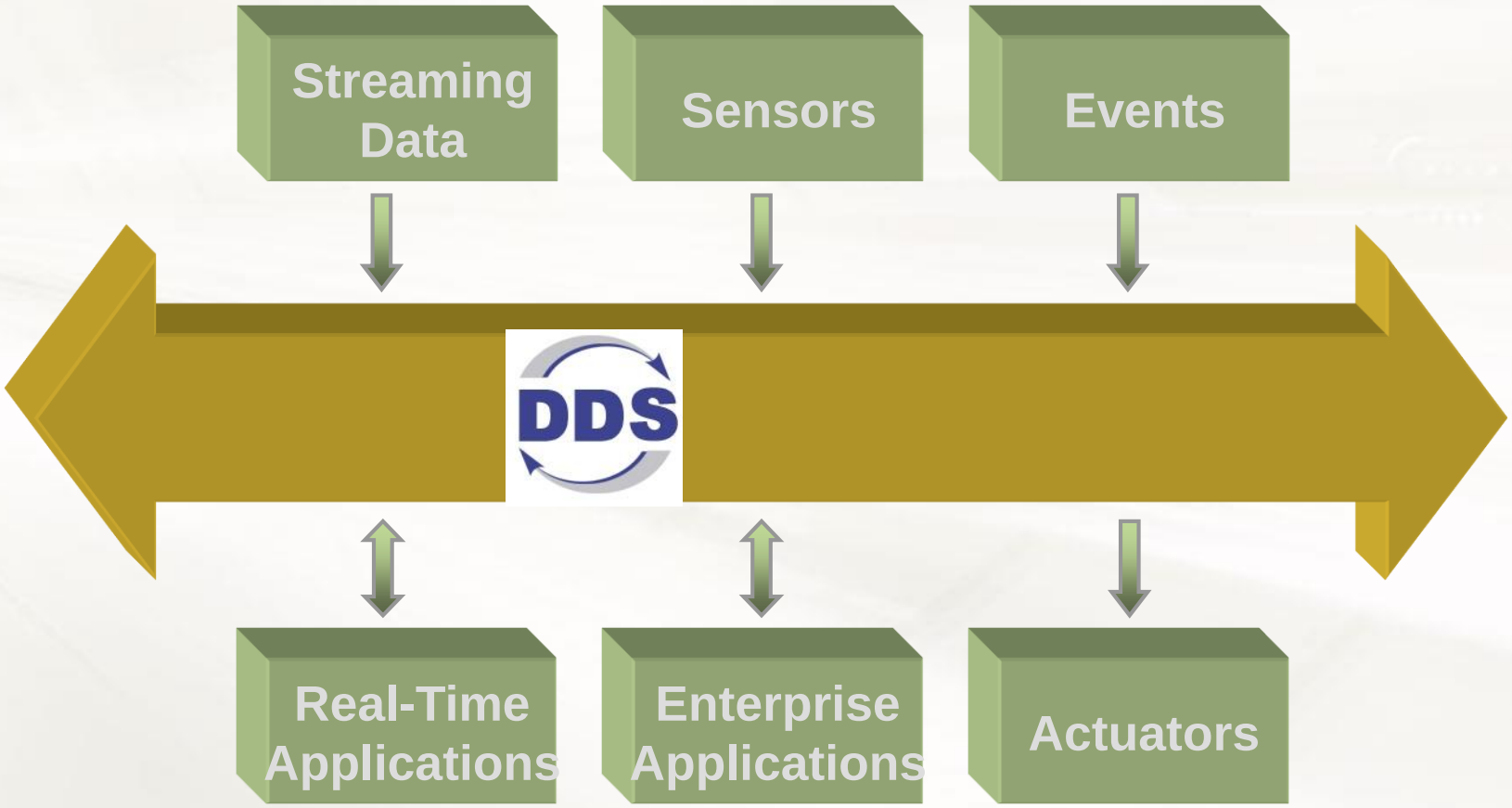
8/16/15, 11:29 PM

Jonas Boner: One of the authors of the Reactive Manifesto

Data Distribution Service (DDS)

Reactive Middleware

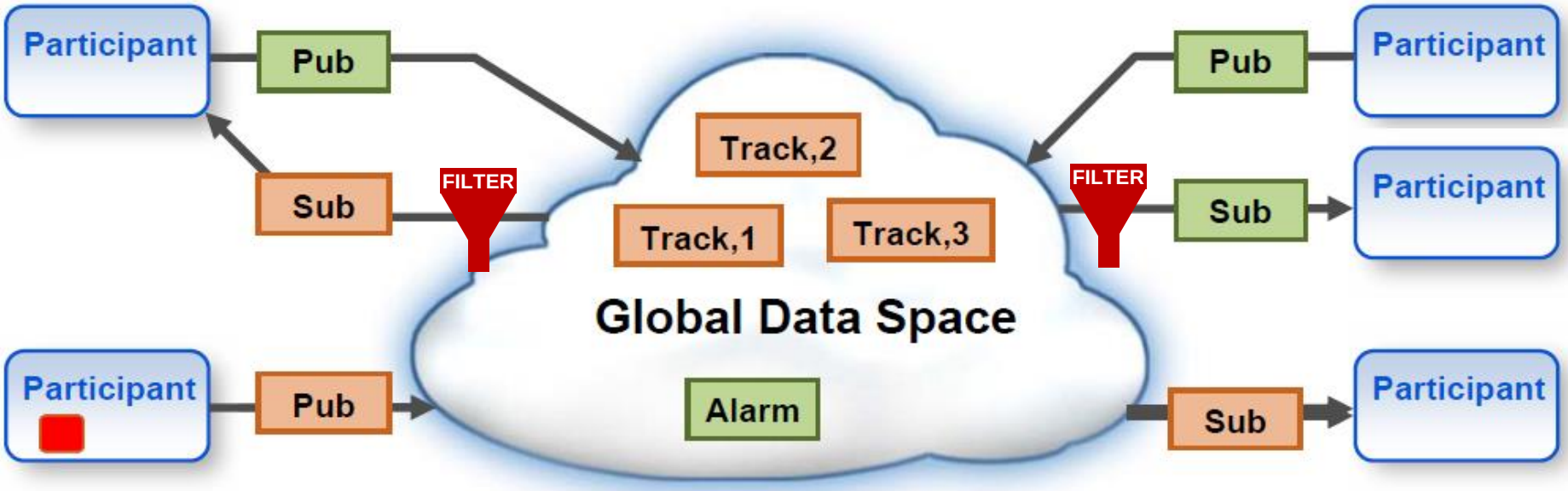
DDS: Data Connectivity Standard for the Industrial IoT



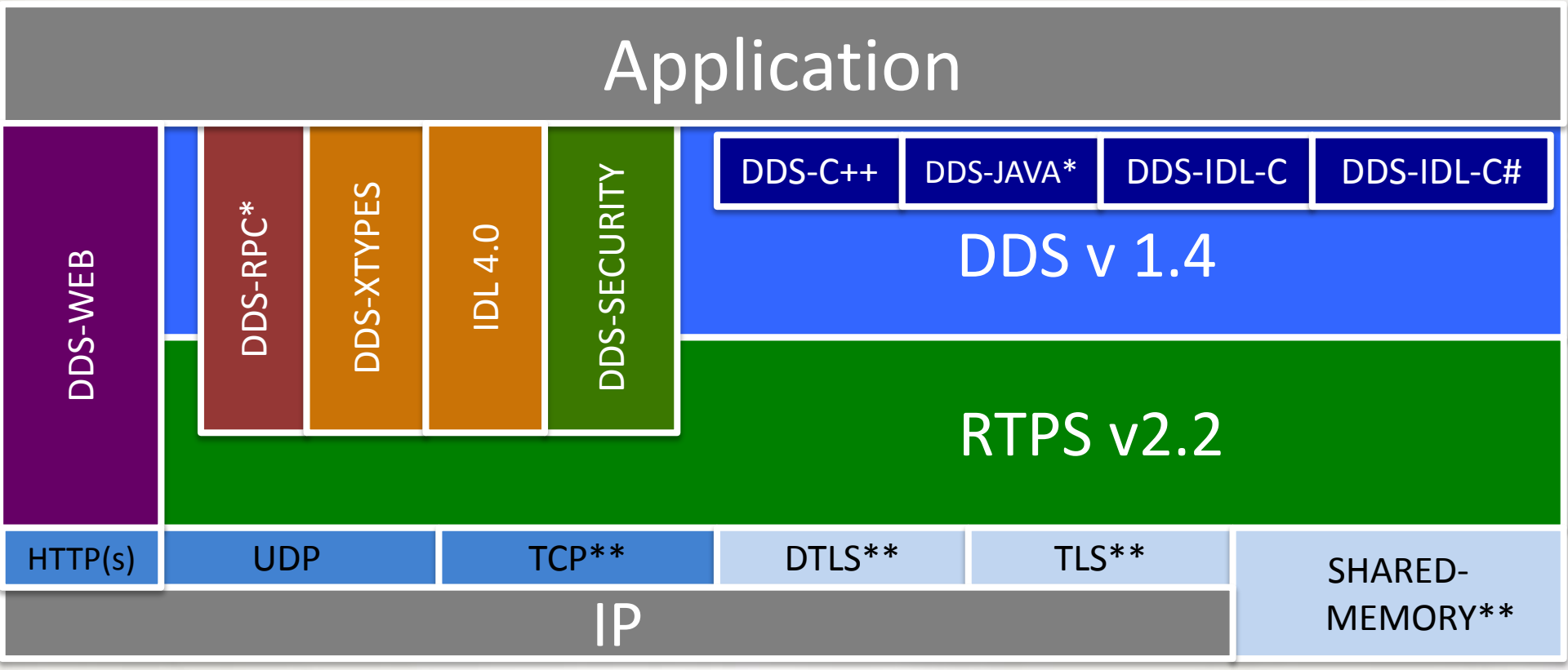
DDS Communication Model

Provides a "**Global Data Space**" that is accessible to all interested applications.

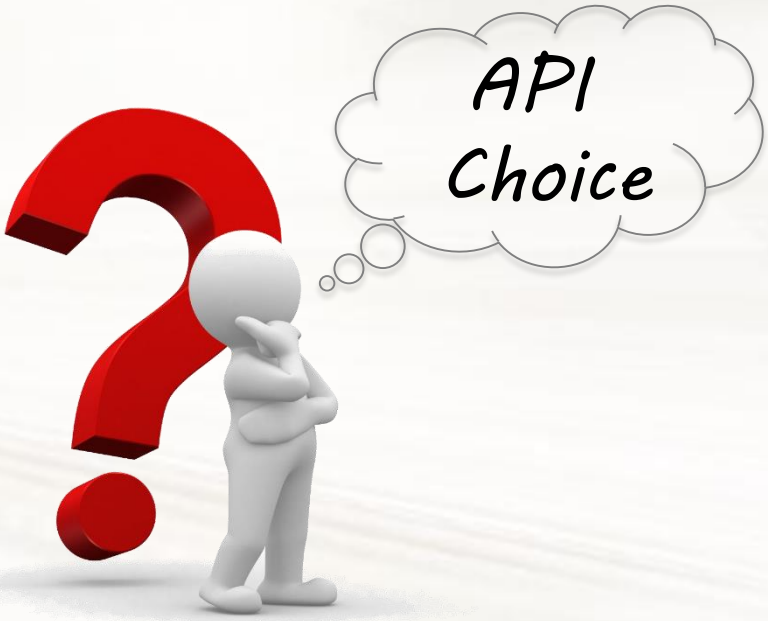
- Data objects addressed by **Domain, Topic** and **Key**
- Subscriptions are **decoupled** from Publications
- Contracts established by means of **QoS**
- Automatic **discovery** and **configuration**



The DDS Standard Family



STANDARD



AWESOME

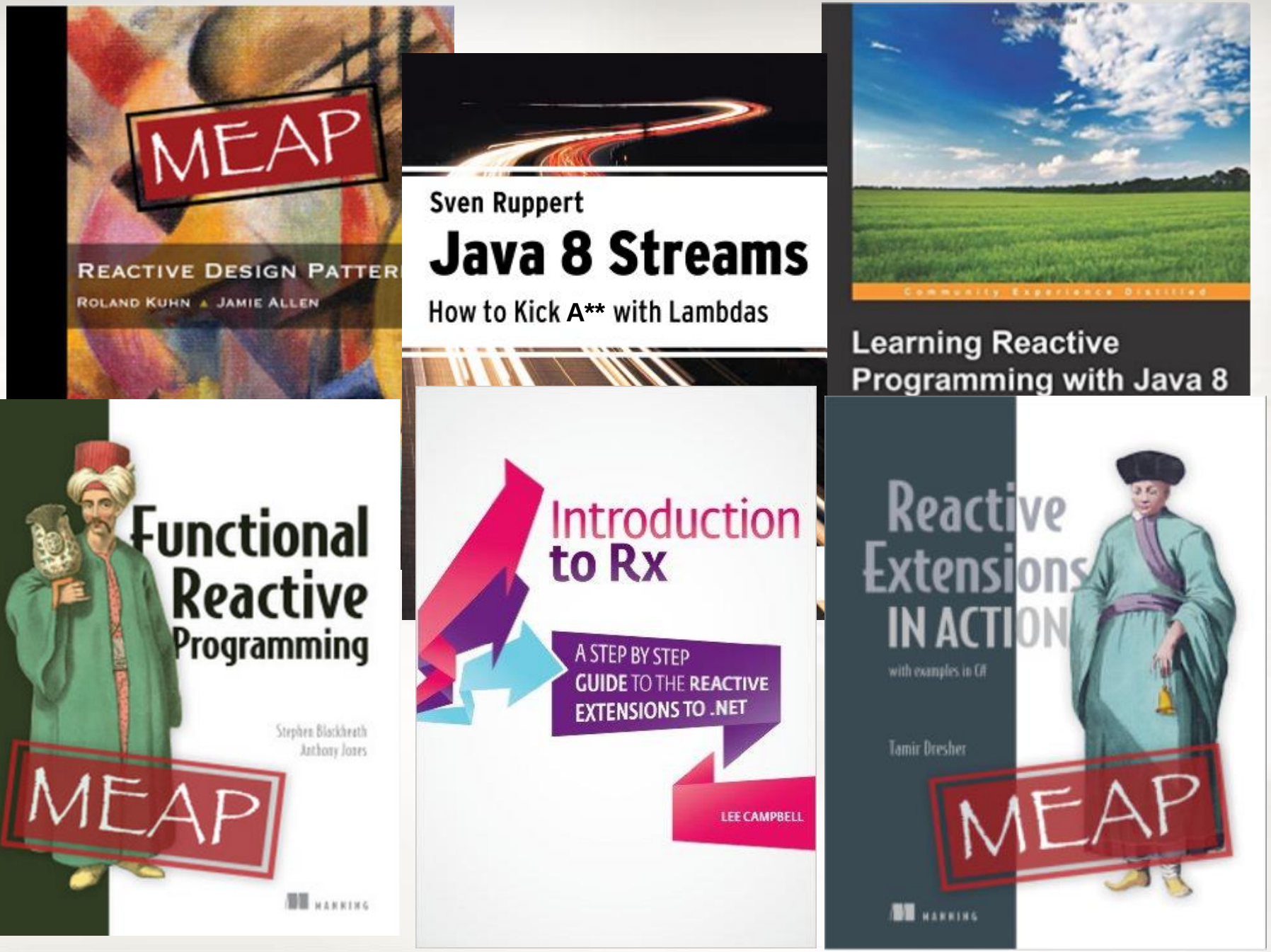


Image source: Amazon, Google

Stream Processing

An architectural style that operates on a continuous sequence of data.

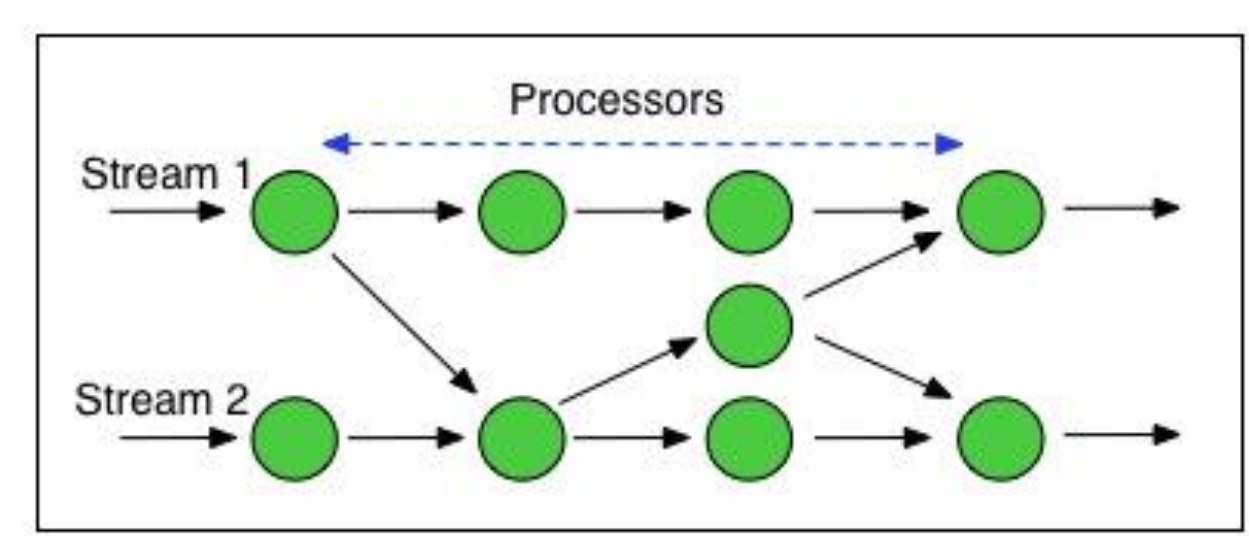
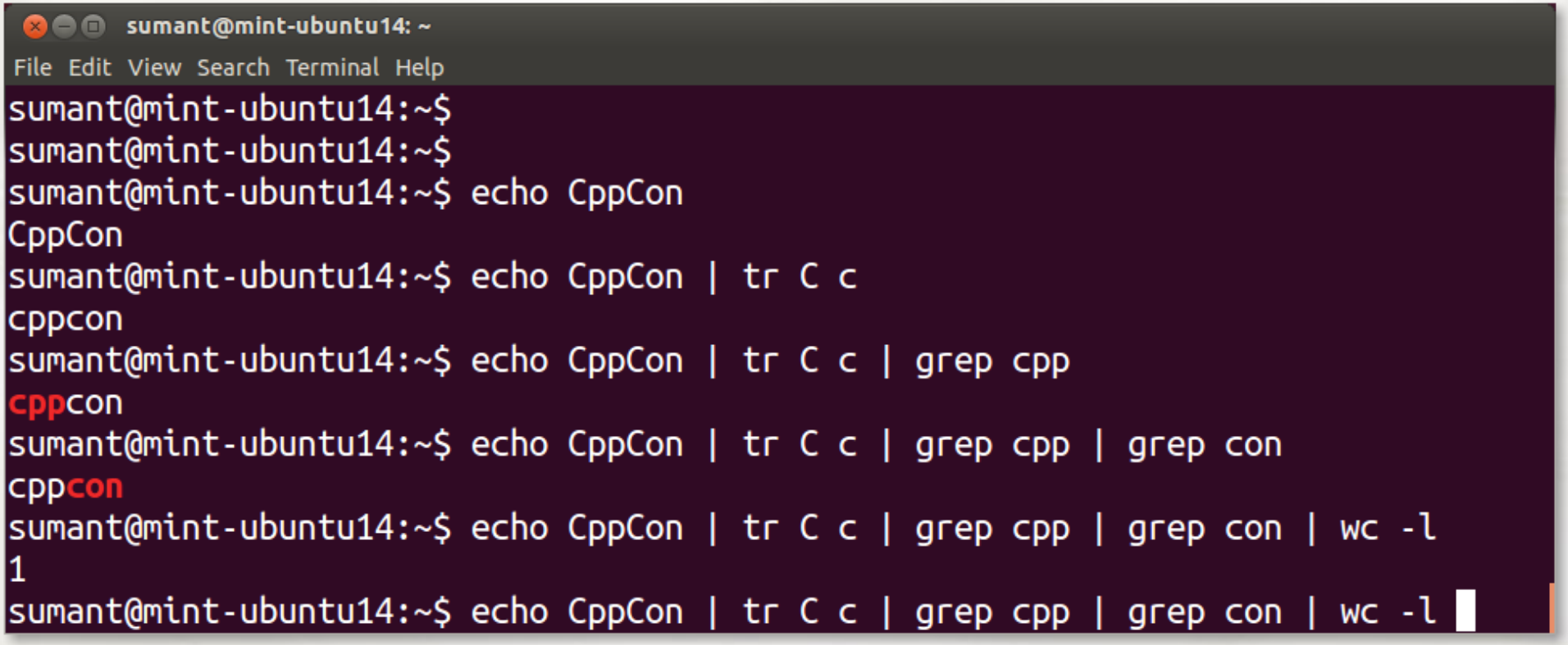
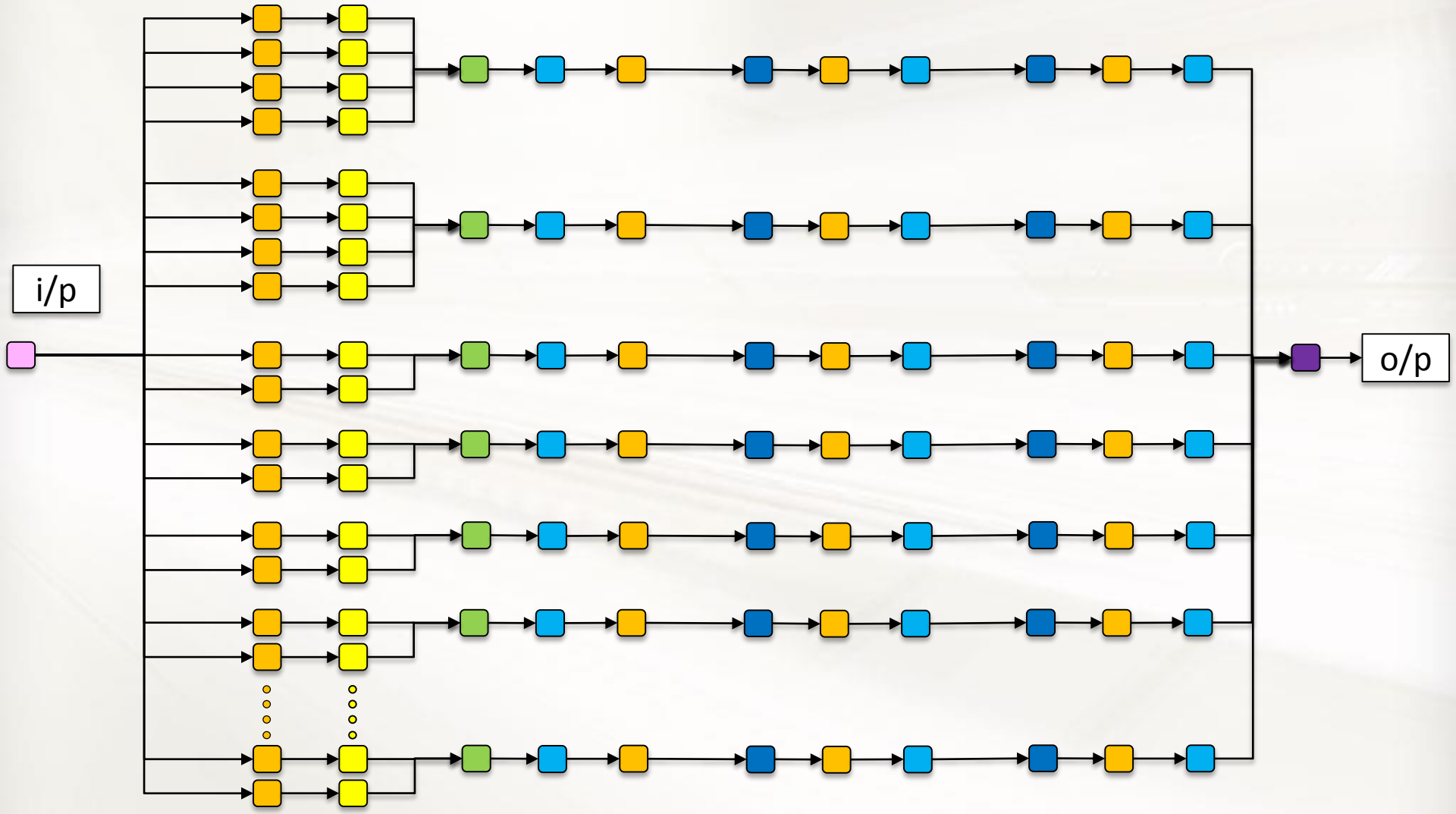


Image credit: eecs.berkeley.edu

Unix command line (pipes and filter)



Data Pipelines



Where Once CombineLatest Select Scan Merge Raw Data

Reactive Extensions (Rx)

- Invented at Microsoft (Erik Meijer and team)
- API for composing asynchronous and event-based programs using **observable** streams
- Rx = Observables + Composition + Schedulers
- Observables: First-class streams
- Composition: Filter, select, aggregate (reduce), compose and perform time-based operations on multiple streams
- Schedulers: Concurrency. Thread-pools, etc.
- Uses **Functional Programming** (Monadic)
- Rx.NET, RxJava, RxJS, RxCpp, RxRuby, RxPython, and many more…

More info: https://speakerdeck.com/benjchristensen/reactive-programming-with-rx-at-qconsf-2014

Duality of Push and Pull Interfaces (C#)

Pull	Push

`IEnumerator<out T>`
`  T Current { get; }`
`  bool MoveNext()`
	`IObserver<in T>`
	`  void OnNext(T value)`
	`  void OnCompleted()`
	`  void OnError(Exception ex)`

`IEnumerable<out T>` `  IEnumerator<T> GetEnumerator();`	`IObservable<out T>` `  IDisposable Subscribe(IObserver<T>)`

Duality (video):
http://channel9.msdn.com/Shows/Going+Deep/Expert-to-Expert-Brian-Beckman-and-Erik-Meijer-Inside-the-NET-Reactive-Framework-Rx

Sync/Async, One/Many

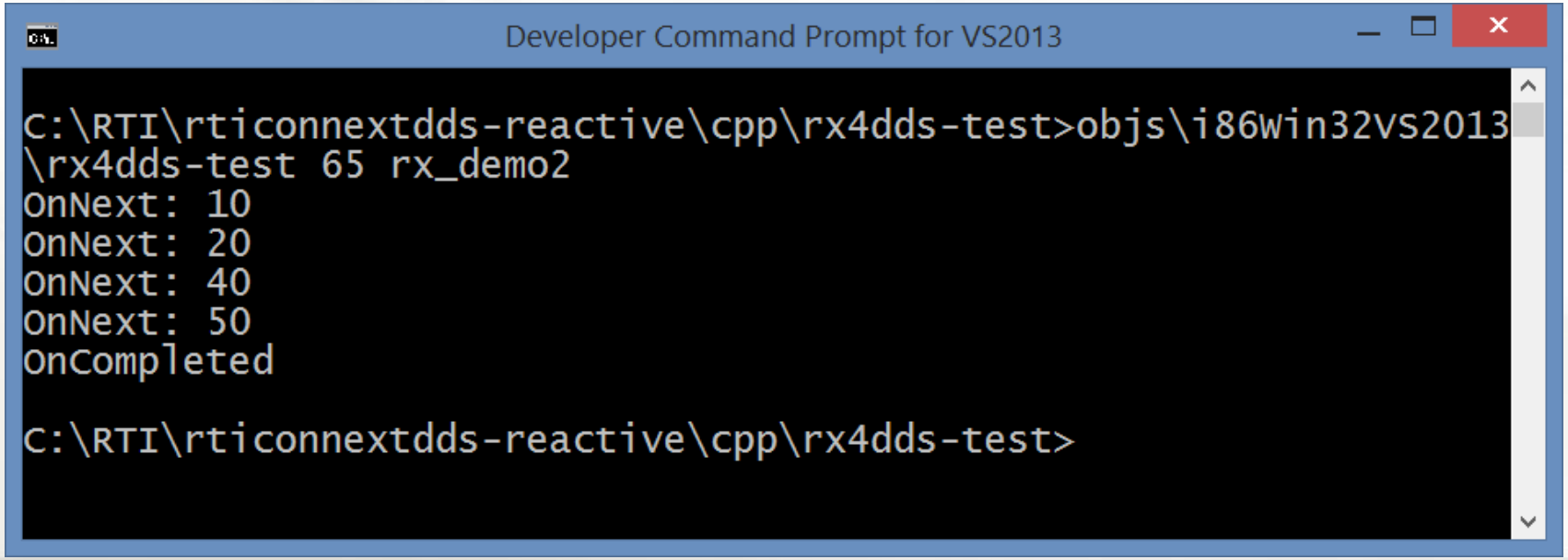
	Single	Multiple
Sync	`T get_data()`	`std::vector<T>::iterator`
Async	`future<T> with .then() .next()` (hpx, boost, C++17)	`rxcpp::observable<T>`

RxCpp Example (1/3)

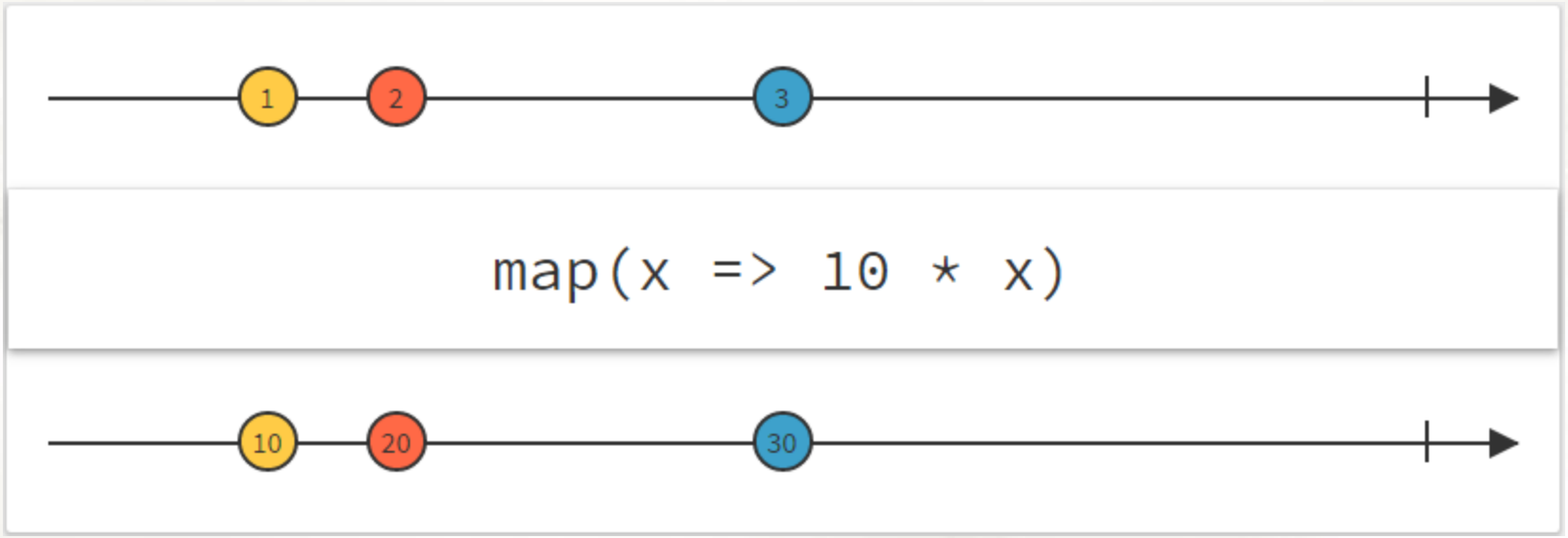
```
rxcpp::observable<int> values =
  rxcpp::observable<>::range(1, 5);

auto subscription = values.subscribe(
  [](int v){ printf("OnNext: %d\n", v); },
  [](){ printf("OnCompleted\n"); });
```

RxCpp Example (2/3)

```
auto subscription =
  rxcpp::observable<>::range(1, 5)
    .map([](int i) { return 10*i; })
    .filter([](int v) { return (v % 15) != 0; })
    .subscribe(
      [](int v){ printf("OnNext: %d\n", v); },
      [](){ printf("OnCompleted\n"); });
```



Map Marble Diagram

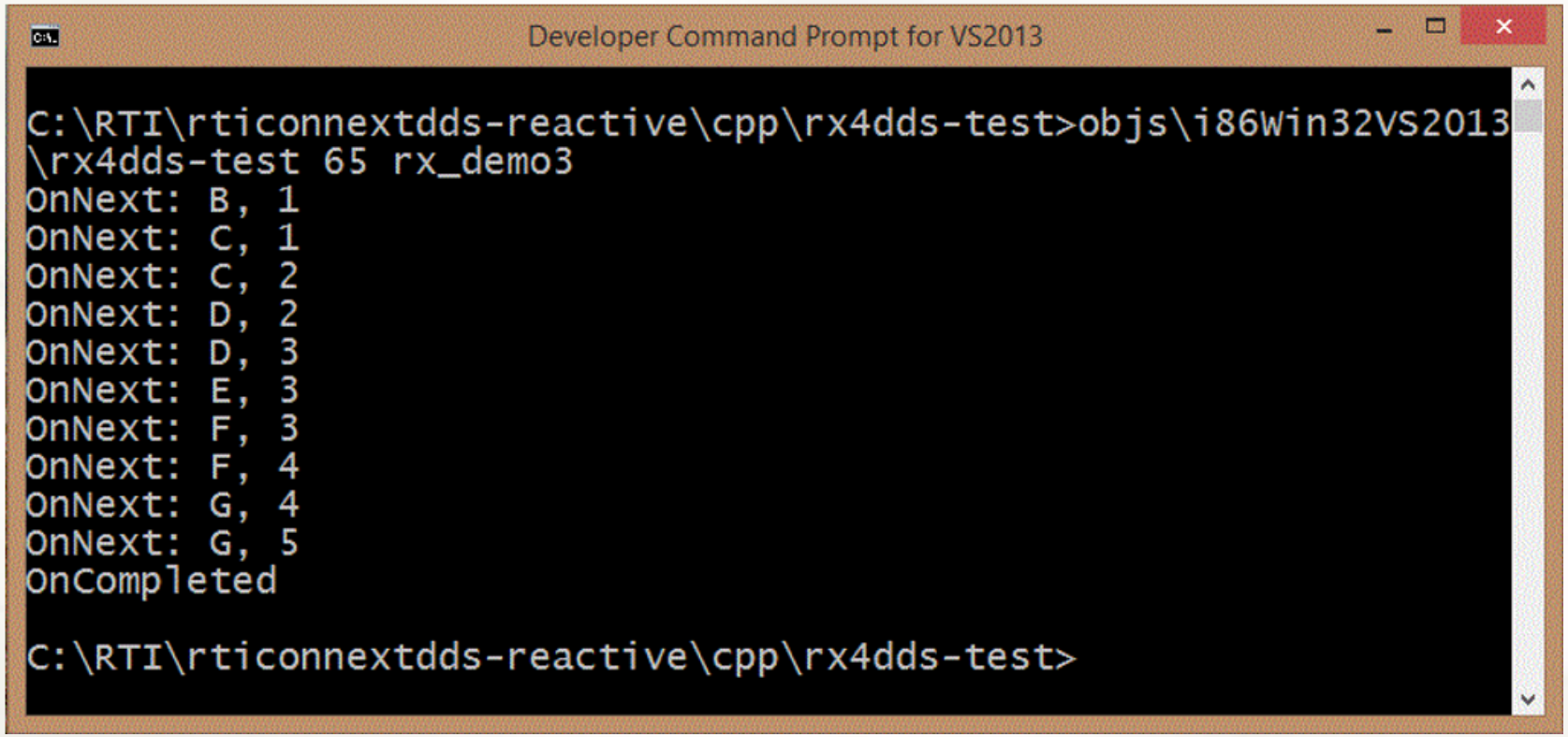


Credit: rxmarbles.com

RxCpp Example (3/3)

```
auto o1 =
  rx::observable<>::interval(std::chrono::seconds(2));
auto o2 =
  rx::observable<>::interval(std::chrono::seconds(3));

auto values = o1.map([](int i) { return char('A' + i); })
                .combine_latest(o2);

values
  .take(10)
  .subscribe(
      [](std::tuple<char, int> v) {
        printf("OnNext: %c, %d\n",
               std::get<0>(v), std::get<1>(v));
      },
      []() { printf("OnCompleted\n"); });
```

RxCpp Output



Rx4DDS

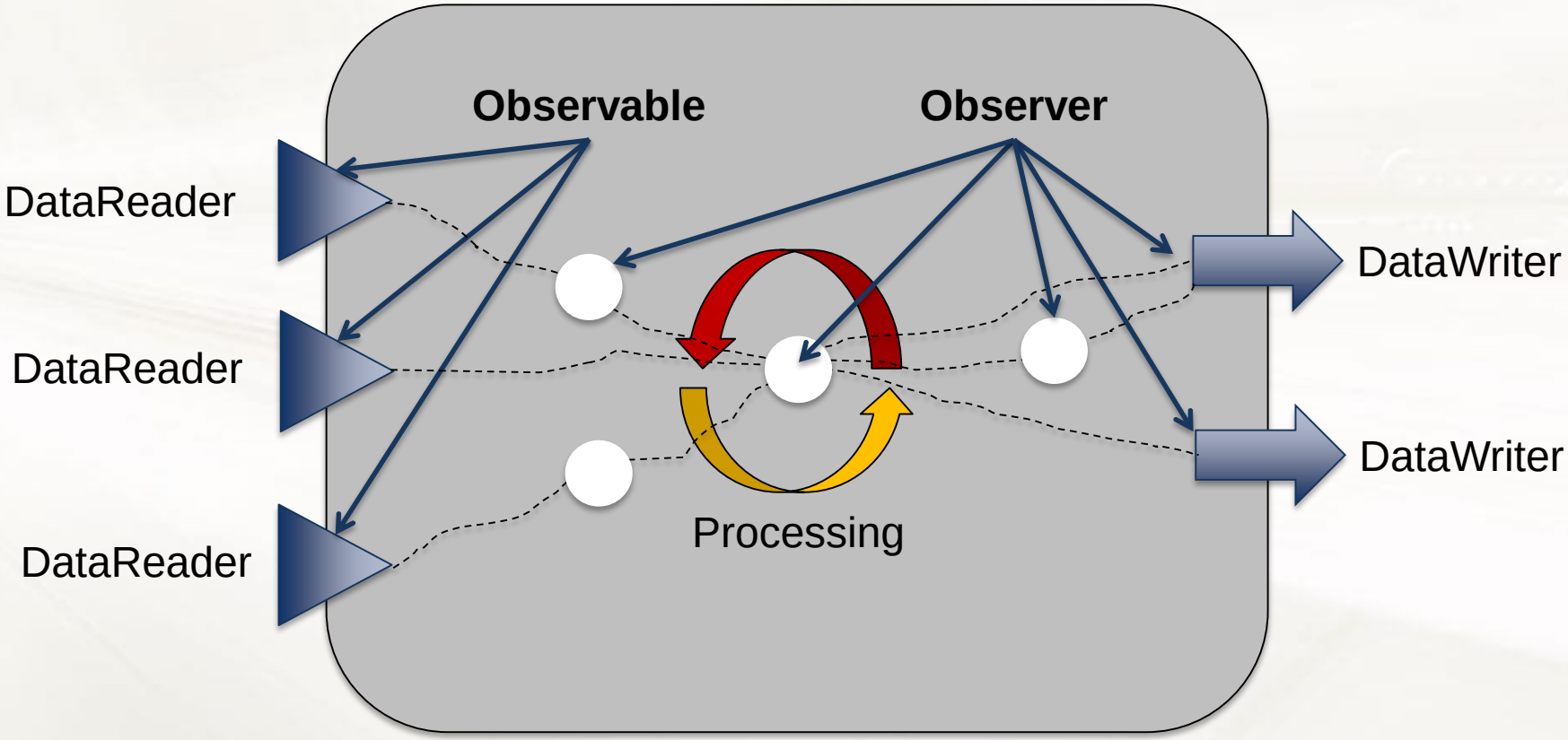
Reactive Stream Processing in IIoT

Rx4DDS = DDS + Rx

- Rx bindings for DDS data
  - In C++11, C#, and JavaScript
- Anything that produces data is an Observable
  - Topics
  - Discovery
  - Statuses

More info: http://rticommunity.github.io/rticonnextdds-reactive/

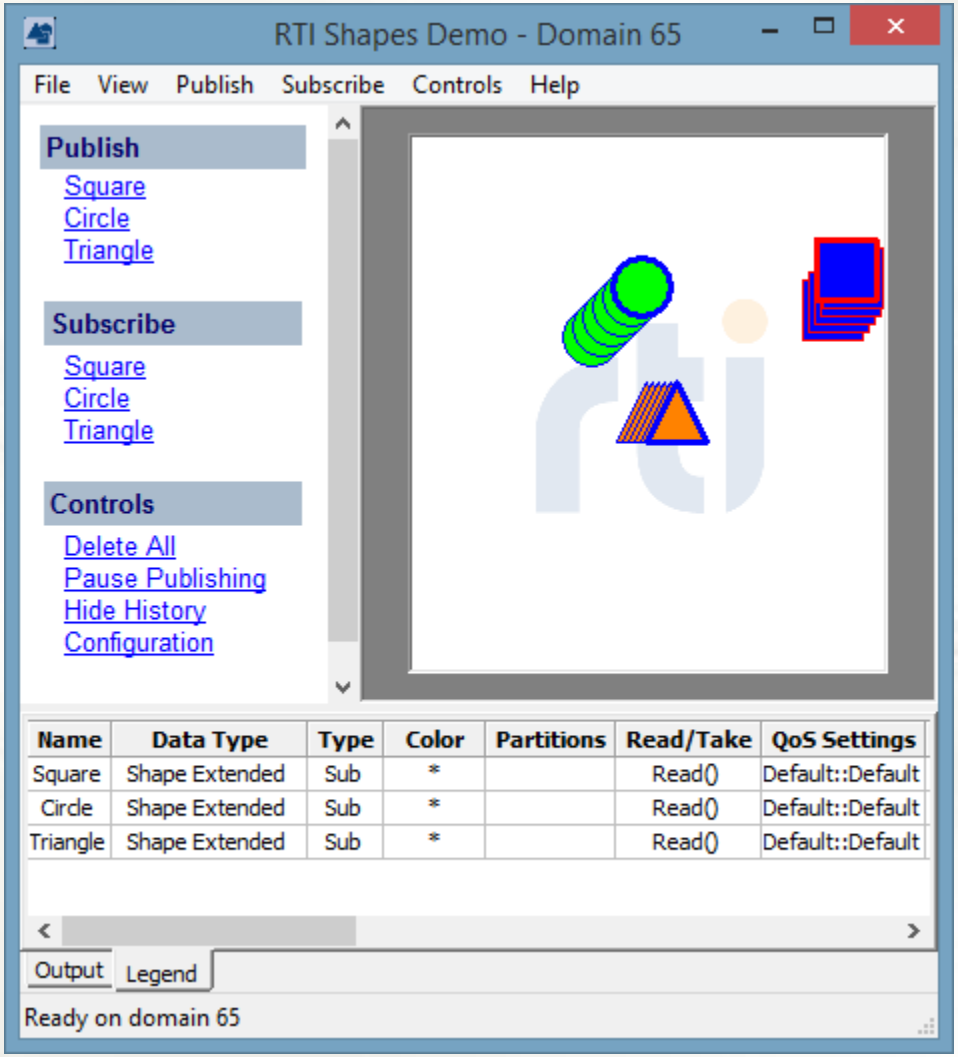
DDS for Distribution, Rx for Processing



DDS and Rx: A Great Match

	DDS	Rx
Design Methodology	Data-Centric	Reactive, Compositional
Architecture	Distributed Publish-subscribe	In-memory Observable-Observer. Monadic
Anonymity/loose coupling	Publisher does not know about subscribers	Observable does not know about observers
Data Streaming	A Topic<T> is a stream of data samples of type T	An Observable<T> is a stream of object of type T
Stream lifecycle	Instances (keys) have lifecycle (New→Alive→Disposed)	Observables have lifecycle OnNext*[OnCompleted|OnError]
Data Publication	Publisher may publish without any subscriber	"Hot" Observables
Data Reception	Subscriber may read the same data over and over	"Cold" Observables

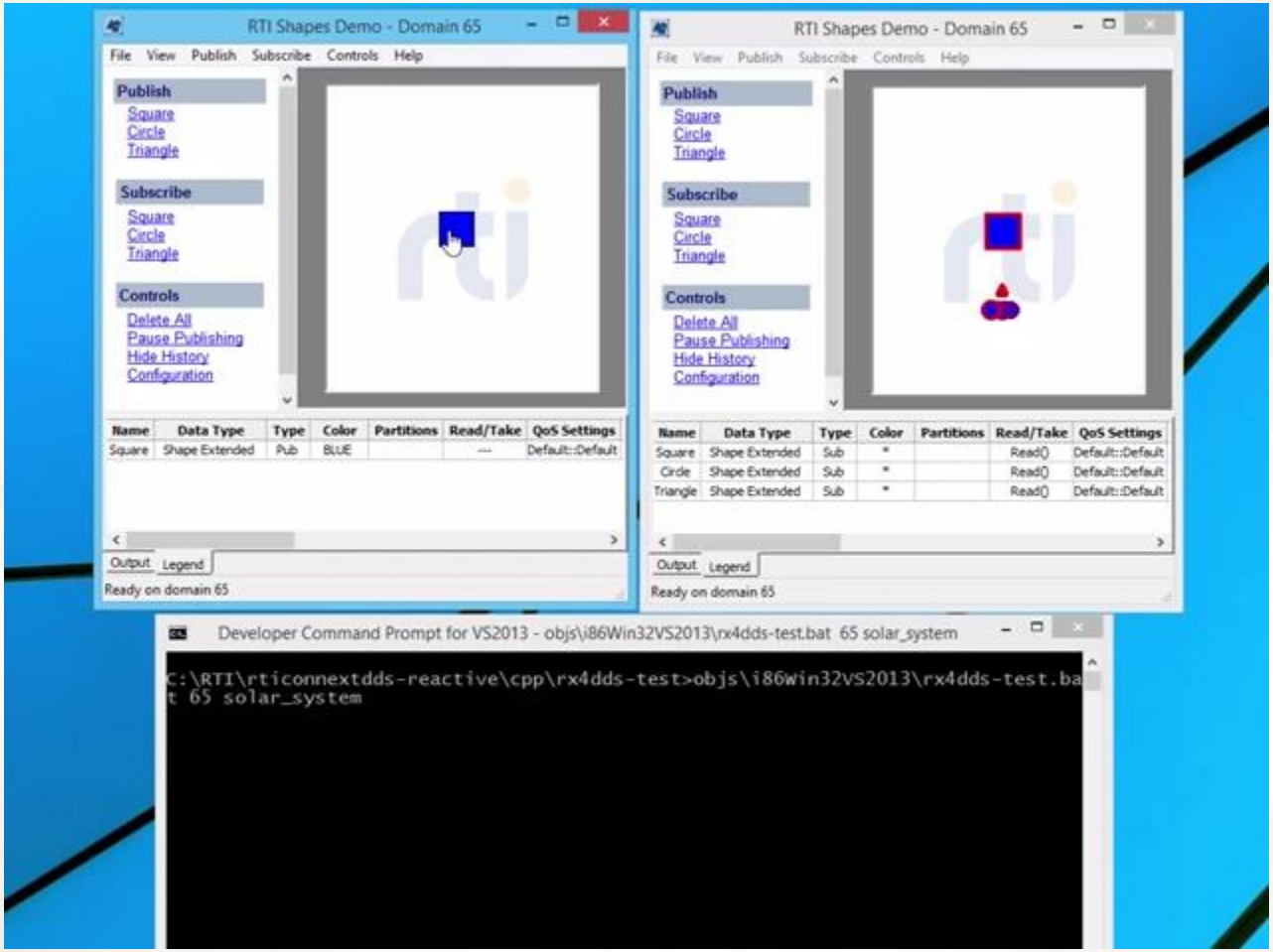
Stream Processing Demos with Shapes



- 3 DDS Topics
  - "Square"
  - "Triangle"
  - "Circle"

```
class ShapeType
{
   string color; //@key
   int shapesize;
   int x;
   int y;
}
```

Solar System Demo



Link: https://youtu.be/mHNyEPeOPHg (1 min)

```
rx4dds::TopicSubscription<ShapeType>
   topic_sub(participant, "Square", waitset, worker);

rx::observable<LoanedSample<ShapeType>> source =
    topic_sub.create_observable();

rx::observable<ShapeType> square_track =
    source >> rx4dds::complete_on_dispose()
           >> rx4dds::error_on_no_alive_writers()
           >> filter([](LoanedSample<ShapeType> s) {
                 return s.info().valid();
              }) // skip invalid samples
           >> map([](LoanedSample<ShapeType> valid) {
                 return valid.data();
              }); // map samples to data
```

Map to Circle Track

```
int circle_degree = 0;

rx::observable<ShapeType> circle_track =
square_track
    .map([circle_degree](ShapeType & square) mutable
    {
      circle_degree = (circle_degree + 3) % 360;
      return shape_location(square, circle_degree);
    })
    .tap([circle_writer](ShapeType & circle) mutable {
            circle_writer.write(circle);
    }); // tap replaced as publish_over_dds later
```

Map to Triangle Track

```
int tri_degree = 0;

rx::observable<ShapeType> triangle_track =
circle_track
    .map([tri_degree](ShapeType & circle) mutable
    {
      tri_degree = (tri_degree + 9) % 360;
      return shape_location(circle, tri_degree);
    })
    >> rx4dds::publish_over_dds(triangle_writer);


triangle_track.subscribe();
```

Naming Transformations (C++14)

```
auto skip_invalid_samples()
{
  return [](auto src_observable) {

    return src_observable
             .filter([](auto & sample) {
                       return sample.info().valid()
             });

  };
}
```

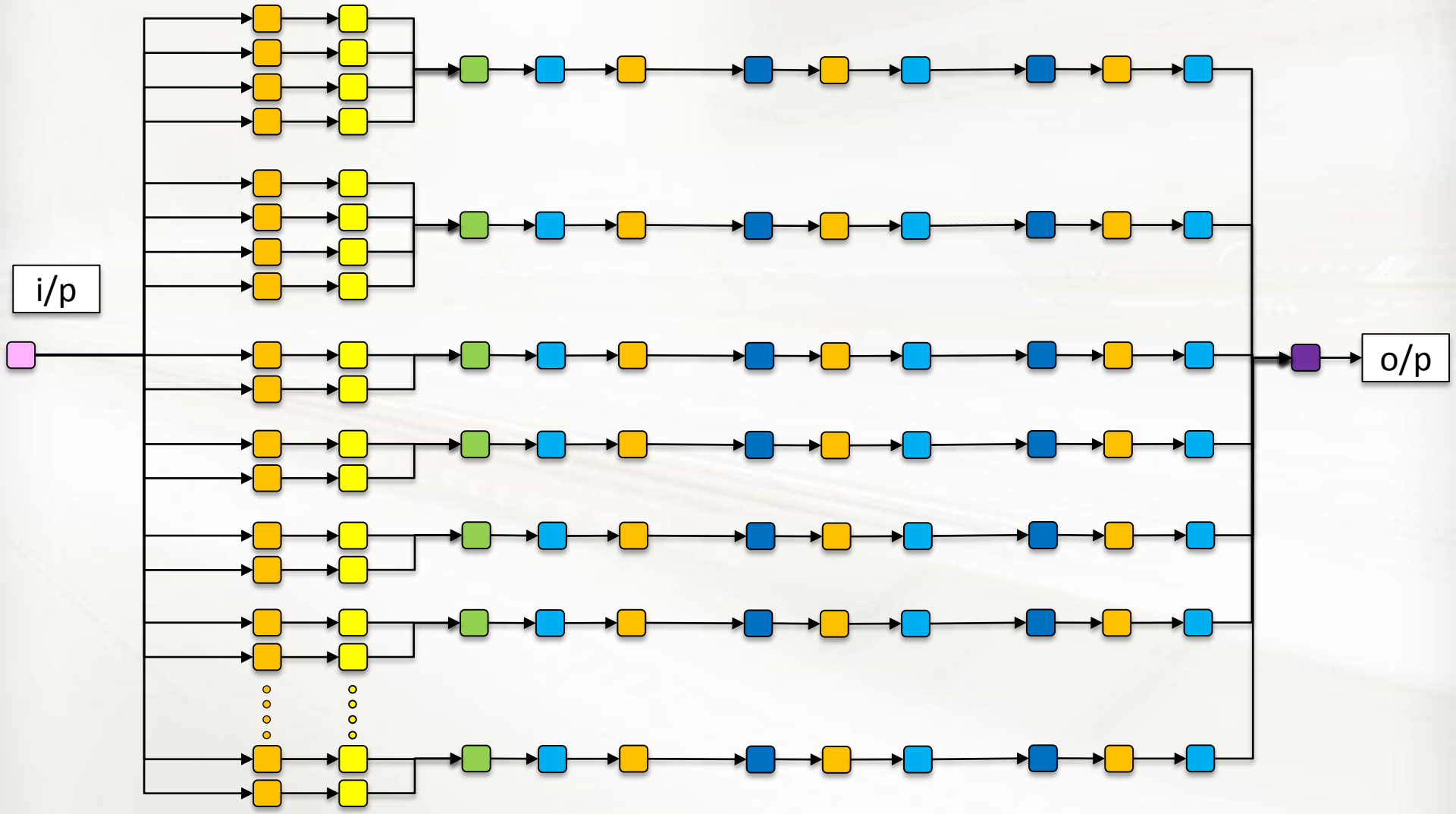
Naming Transformations (C++11)

```
struct MapSampleToDataOp
{
  template <class Observable>
  rx::observable<typename Observable::value_type::DataType>
  operator ()(Observable src) const
  {
      typedef typename Observable::value_type LoanedSample;
      return src.map([](LoanedSample & sample) {
        return sample.data()
      });
  }
};

inline MapSampleToDataOp map_samples_to_data()
{
    return MapSampleToDataOp();
}
```

27,572 cables

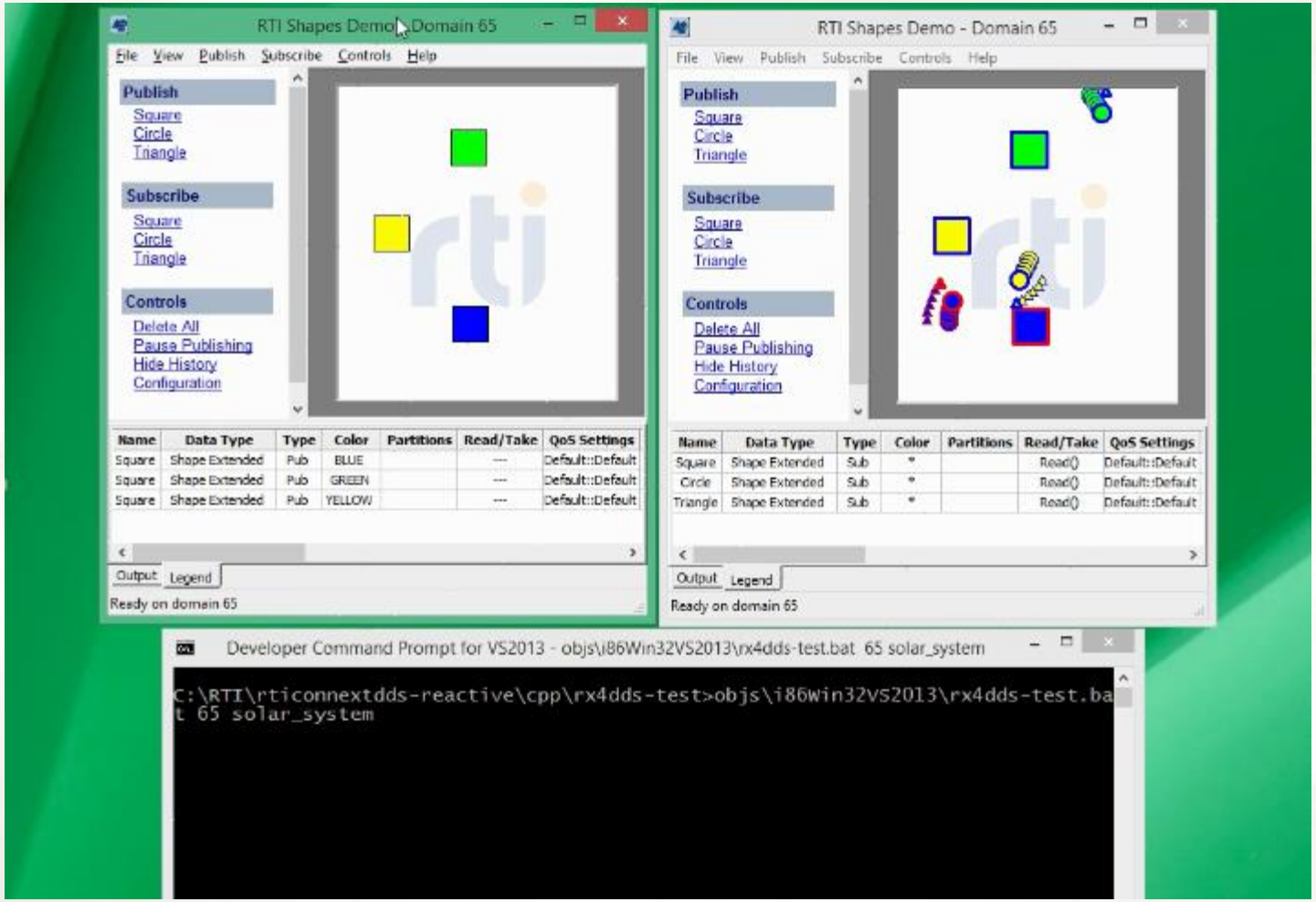
Source: Google Images

Data Pipelines Demo Next



Where Once CombineLatest Select Scan Merge Raw Data

Data Pipelines Demo



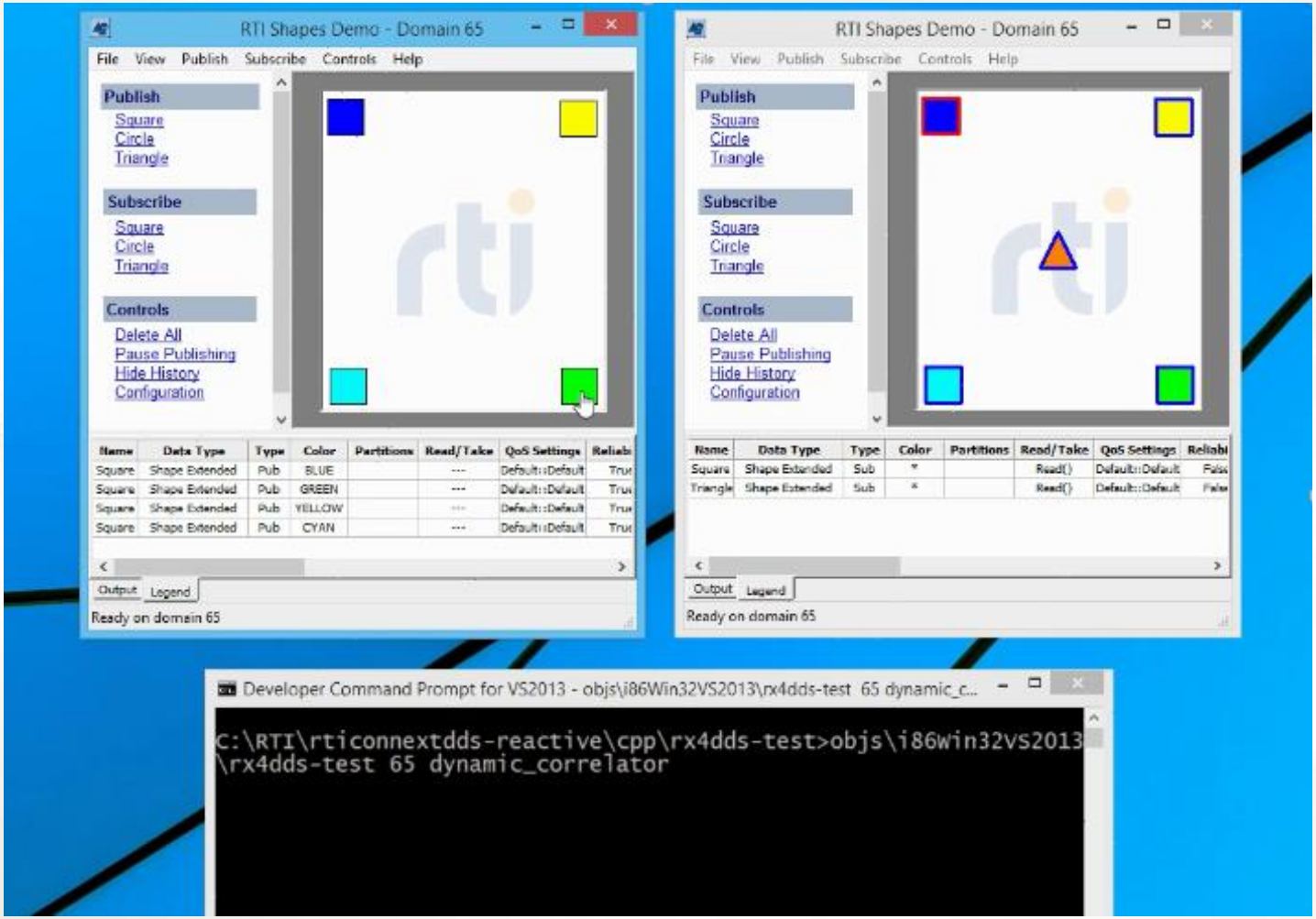
Link: https://youtu.be/2Pz91e7yR5I (3 min)

```
rx4dds::TopicSubscription<ShapeType>
   topic_sub(participant, "Square", waitset, worker);

auto grouped_stream =
  topic_sub.create_observable()
      >> group_by_instance ([](ShapeType & shape) {
            return shape.color();
         });
```

```
decltype(grouped_stream) ===
  rx::observable<
     rx::grouped_observable<
        string, LoanedSample<ShapeType>
     >
  >
```

```
grouped_stream
  .flat_map([circle_writer, triangle_writer]
            (GroupedShapeObservable go) {
    rx::observable<ShapeType> inner_transformed =
        go >> to_unkeyed()
           >> complete_on_dispose()
           >> error_on_no_alive_writers()
           >> skip_invalid_samples()
           >> map_samples_to_data()
           >> map_to_circle_track() // as shown before
           >> publish_over_dds(
                 circle_writer, ShapeType(go.key())
           >> map_to_triangle_track() // as shown before
           >> publish_over_dds(
                 triangle_writer, ShapeType(go.key());
    return inner_transformed;
  }).subscribe();
```

Dynamic Correlator



Link: https://youtu.be/tZutExU6r0w (1 min)

```
grouped_stream
  .map([](GroupedShapeObservable go) {
    return go >> rx4dds::to_unkeyed()
              >> rx4dds::complete_on_dispose()
              >> rx4dds::error_on_no_alive_writers()
              >> rx4dds::skip_invalid_samples()
              >> rx4dds::map_samples_to_data();
  })
  >> rx4dds::coalesce_alive()
  >> map([](const vector<rxcpp::observable<ShapeType>> & srcs) {
       return rx4dds::combine_latest(srcs);
     })
  >> rxcpp::switch_on_next()
  >> rxcpp::map([](const std::vector<ShapeType> & shapes) {
         return calculate_average(shapes);
     })
 >> rx4dds::publish_over_dds(triangle_writer, ShapeType("ORANGE"));
```

My ¢2 on FP in C++11

- Noisy!!
  - Lambda captures, `mutable`
- Lambdas don't extend local object lifetimes when captured
  - `Consequence: shared_ptr` galore!
- Death by `auto`
  - auto-heavy code is not readable
  - Compiler errors far away from the erroneous line
  - Please don't remove types if you have them already
  - Types are the best documentation
- RxCpp
  - Learning Rx.NET, RxJS first strongly recommended
  - Frequently very, very long observable types
  - Use `as_dynamic()` but beware of performance implications

Resources

- Rx
  - LearnRx (RxJS) by JHusain from Netflix
  - Intro To Rx (Rx.NET) by Lee Campbell
  - rxmarbles.com (interactive marble diagrams)
  - ReactiveX.io (API docs in multiple languages)
- Rx4DDS Stream Processing
  - Github [bit.ly/cppcon-rx4dds]
  - Research Paper [link]
- DDS
  - Object Management Group [http://portals.omg.org/dds]
  - RTI [www.rti.com]

Thank You!

Extra Slides

Hot and Cold Observables

	Hot	Cold
Meaning	Emits whether you are ready or not and regardless of subscribers	Emits when requested. Starts from the beginning with every subscription
Examples	1. Mouse movements 2. User input 3. Stock prices 4. DDS Topic	1. Reading a file 2. Database query 3. `observable.interval`
Semantics	Receiver must keep up or use sampling. May lose data	Receiver can exert backpressure

Decoupling

- Separating production of data from consumption
- Two ways to produce (access) DDS data
  - WaitSet-based blocking and dispatch (like select)
  - Listener-based (m/w calls you back)
- Each has pros/cons and different APIs and leads to very different code structure
- Observables allows us to change data production technique without changing data consuming code
- Data consuming code is written in the same declarative chaining style

DDS and Rx: A Great Match

DDS Concept	Rx Concept/Type/Operator
Topic of type T	An object that implements `IObservable<T>`, which internally creates a `DataReader<T>`
Communication status, Discovery event streams	`IObservable<SampleLostStatus>` `IObservable<SubscriptionBuiltinTopicData>`
Topic of type T with key type=Key	`IObservable<IGroupedObservable<Key, T>>`
Detect a new instance	Notify Observers about a new `IGroupedObservable<Key, T>` with key==instance. Invoke `IObserver<IGroupedObservable<Key, T>>.OnNext()`
Dispose an instance	Notify Observers through `IObserver<IGroupedObservable<Key,T>>.OnCompleted()`
Take an instance update of type T	Notify Observers about a new value of T using `Iobserver<T>.OnNext()`
Read with history=N	`IObservable<T>.Replay(N)` (Produces a new `IObservable<T>`)

DDS and Rx: A Great Match

DDS Concept	Rx Concept/Type/Operation
Query Conditions	`Iobservable<T>.Where(…) OR` `Iobservable<T>.GroupBy(…)`
SELECT in CFT expression	`IObservable<T>.Select(...)`
FROM in CFT expression	`DDSObservable.FromTopic("Topic1")` `DDSObservable.FromKeyedTopic("Topic2")`
WHERE in CFT expression	`IObservable<T>.Where(...)`
ORDER BY in CFT expression	`IObservable<T>.OrderBy(...)`

MultiTopic (INNER JOIN)	`IObservable<T>.Join(...)`
	`.Where(...)`
	`.Select(...)`

Join between DDS and non-DDS data	`Join, CombineLatest, Zip`