# Outline Thesis Feedback Control and Backpressure

* Title page
  + Title of the thesis
  + Date of the thesis defense
  + Name and student number
  + Chosen specialization
  + Master program
  + Faculty EEMCS, TU Delft
* Preface
  + Explain the topic and context (institute/company)
  + Main findings in a few lines
  + Names of the members of the thesis committee
  + Acknowledgements
  + Finish with ‘name’ and ‘date’
* Introduction
  + Main research themes
  + Overview of the report
* Previous work
  + Relevant scientific publications
  + Reformulates the research questions to comply with the state of the art in the field
* Performed work
  + Several chapters
* Discussion of the results
  + Discussion
  + Conclusion
  + Recommendations and suggestions for future research
* Appendices
  + Scientific paper written about this work

# Actual content

* Problem statement
  + Reactive vs. Interactive
  + Rx
    - Core components
    - Derivation
    - Operators
    - Hot/cold
    - Subjects
  + Backpressure
    - What is it?
    - Why does it occur?
  + When does it become a problem
    - General problem
      * …
      * Hot/cold observables
    - Examples of backpressure being a problem
      * [Fast producer, slow consumer]
      * Zip
      * …
  + Different approaches to solve this
    - Reactive programming
      * <https://xgrommx.github.io/rx-book/content/getting_started_with_rxjs/creating_and_querying_observable_sequences/backpressure.html>
      * <https://github.com/ReactiveX/RxJava/wiki/Backpressure>
      * Lossy (packets being dropped)
        + Debounce (throttleWithTimeout)
        + Throttling (throttleFirst)
        + Sampling (throttleLast)
        + ~~Pausable Observable in RxJS~~
      * Loss-less
        + Buffers and Windows

complex buffer constructs

<https://speakerdeck.com/benjchristensen/reactive-streams-with-rx-at-javaone-2014?slide=81> + <https://www.youtube.com/watch?v=g-ixzOcMDF4> -> min 33

<http://www.infoq.com/presentations/rx-service-architecture> -> min. 30

* + - * + ~~Pausable buffers in RxJS~~
    - Transmission Control Protocol
      * [https://en.wikipedia.org/wiki/Transmission\_Control\_Protocol#Flow\_control](https://en.wikipedia.org/wiki/Transmission_Control_Protocol" \l "Flow_control)
    - Reactive Pull 🡪 ReactiveStreams
      * This goes away from the actual push model
      * Problem with certain operators
      * You can’t use this on certain hot streams, such as UI events and time
      * <http://www.reactive-streams.org/>
      * [http://www.reactivemanifesto.org/glossary#Back-Pressure](http://www.reactivemanifesto.org/glossary" \l "Back-Pressure)
* Backpressure can be solved with feedback control
  + This already is the essence of the solutions above (including Reactive Pull)
  + It is however not clear how the backpressure should be implemented on certain operators
  + Instead we try to apply backpressure on the source alone
  + We use feedback control for this
* Feedback Control
  + Introduction 🡪 reference to blog, Hellerstein and Janert
    - Basics of feedback control
    - We apply feedback control to computer science
    - Difference mathematical approach and our approach (also Hellerstein and Janert)
  + Introduce the ball tracker as a toy example
    - First on 1 dimension; implemented with for-loops etc.
    - Use this throughout the rest of the Feedback Control section
  + Feedback control as ‘working with streams’
    - Since we do feedback control for computer science, we want to come up with an API to create feedback systems
      * A good API does not yet exist
      * Some simple stuff in Python
      * Solution from Peti Koch
    - A component sits in between 2 streams and performs some sort of transformation
    - A component is compositional: connecting components, making feedback loop, zipping 🡪 a feedback system is the same as a component
    - Observation: ***a component is the same as a Mealy Machine***
    - Derive the exact type of a component, starting from a Mealy Machine and using category theory
    - Observation: ***a component is an [Arrow](https://en.wikipedia.org/wiki/Arrow_(computer_science))***
    - Introduce the operators on Component
      * Arrow operators
        + arr :: Arrow a => (b -> c) -> a b c
        + (>>>) :: Arrow a => a b c -> a c d -> a b d
        + first :: Arrow a => a b c -> a (b,d) (c,d)
        + second :: Arrow a => a b c -> a (d,b) (d,c)
        + (\*\*\*) :: Arrow a => a b c -> a d e -> a (b,d) (c,e)
        + (&&&) :: Arrow a => a b c -> a b d -> a b (c,d)
        + lift2A :: Arrow a => (b -> c -> d) -> a e b -> a e c -> a e d
        + loop :: Arrow a => a (b,d) (c,d) -> a b c
      * Concat
      * Zip
      * Feedback
      * *<many RxMobile operators>*
      * Lift and LiftA2 (as generalizing over all operators)
    - Ball tracker example with the API
* Solving backpressure with feedback control
  + General solution
  + Solution to specific examples
  + …
  + Toesturen naar reactive streams
  + Reactive streams buiten operatoren
    - Prima als op zelfde uit komt
    - Niet verweven, maar apart
    - Efficienter
* Future work
  + Backpressure:
    - Push solution – control number of workers with certain metrics. This will presumably also work with hot streams such as UI events and time:
      * Queue length
      * Net queue length change
      * In/out ratio per time unit
  + Feedback control:
    - It’s basically an algorithm to solve ‘control problems’. How is the problem class defined where feedback control can be used as an easy solution?
    - It’s difficult to tune PID controllers. (refer to blog) Would it be possible to use ML for this process?
    - The PID controller originally comes from the field of physics and mechanical/electrical engineering and is considered to be the default controller, given its (mathematical) simplicity and quick response to change. Does this still hold for computer science, since we don’t need/use mathematical models? Or is there an alternative, easier to tune, controller that could be considered as default in computer science?