CS 242: Programming Languages

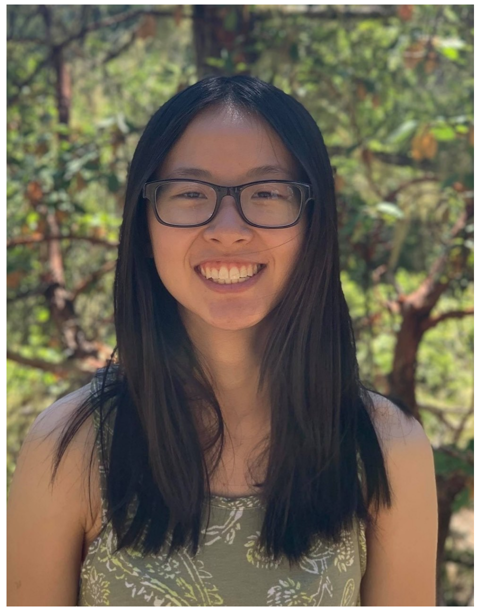
Will Crichton

Course staff

Will

Jayden

Danny


Sicheng

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] * b[i]
  return c
```

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] * b[i]
  return c
```

- **Statements operate on invisible state**
- **Computes word-at-a-time by repetition of assignment/modification**
- **Requires names for arguments, iterator, return value**

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] ∗ b[i]
  return c
```

- Statements operate on invisible state
- Computes word-at-a-time by repetition of assignment/modification
- Requires names for arguments, iterator, return value

```
let innerproduct = zip |> (map ∗) |> (reduce +)
```

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] * b[i]
  return c
```

- **Statements operate on invisible state**
- **Computes word-at-a-time by repetition of assignment/modification**
- **Requires names for arguments, iterator, return value**

```
let innerproduct = zip |> (map *) |> (reduce +)
```

- **Built from composable functions (map, reduce, pipe)**
- **Operates on whole conceptual units (lists), no repeated steps**
- **No names for arguments or temporaries**

```java
public class Person {
    private final String firstName;
    private final String lastName;
    private final Integer age;
    public Person(String firstName,
                  String lastName,
                  Integer age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
    public String getFirst() {
        return firstName;
    }
    public String getLast() {
        return lastName;
    }
    public Integer getAge() {
       return age;
    }
    public Boolean valid() {
       return age > 18;
    }
}
```

David Pollak, “Beginning Scala”

```java
public class Person {
    private final String firstName;
    private final String lastName;
    private final Integer age;
    public Person(String firstName,
                  String lastName,
                  Integer age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
    public String getFirst() {
        return firstName;
    }
    public String getLast() {
        return lastName;
    }
    public Integer getAge() {
       return age;
    }
    public Boolean valid() {
       return age > 18;
    }
}
```

```java
public static List<String> validByAge(List<Person> in) {
   List<Person> people = new ArrayList<Person>();
   for (Person p: in) {
     if (p.valid()) people.add(p);
   }

   Collections.sort(people, new Comparator<Person>() {
      public int compare(Person a, Person b) {
        return a.age() - b.age();
      }
   } );

   List<String> ret = new ArrayList<String>();
   for (Person p: people) {
     ret.add(p.first);
   }
   return ret;
}

List<Person> input = new ArrayList<Person>();
input.add(new Person("John", "Valid", 32));
input.add(new Person("John", "InValid", 17));
input.add(new Person("OtherJohn", "Valid", 19));

List<Person> output = validByAge(input)
```

David Pollak, "Beginning Scala"

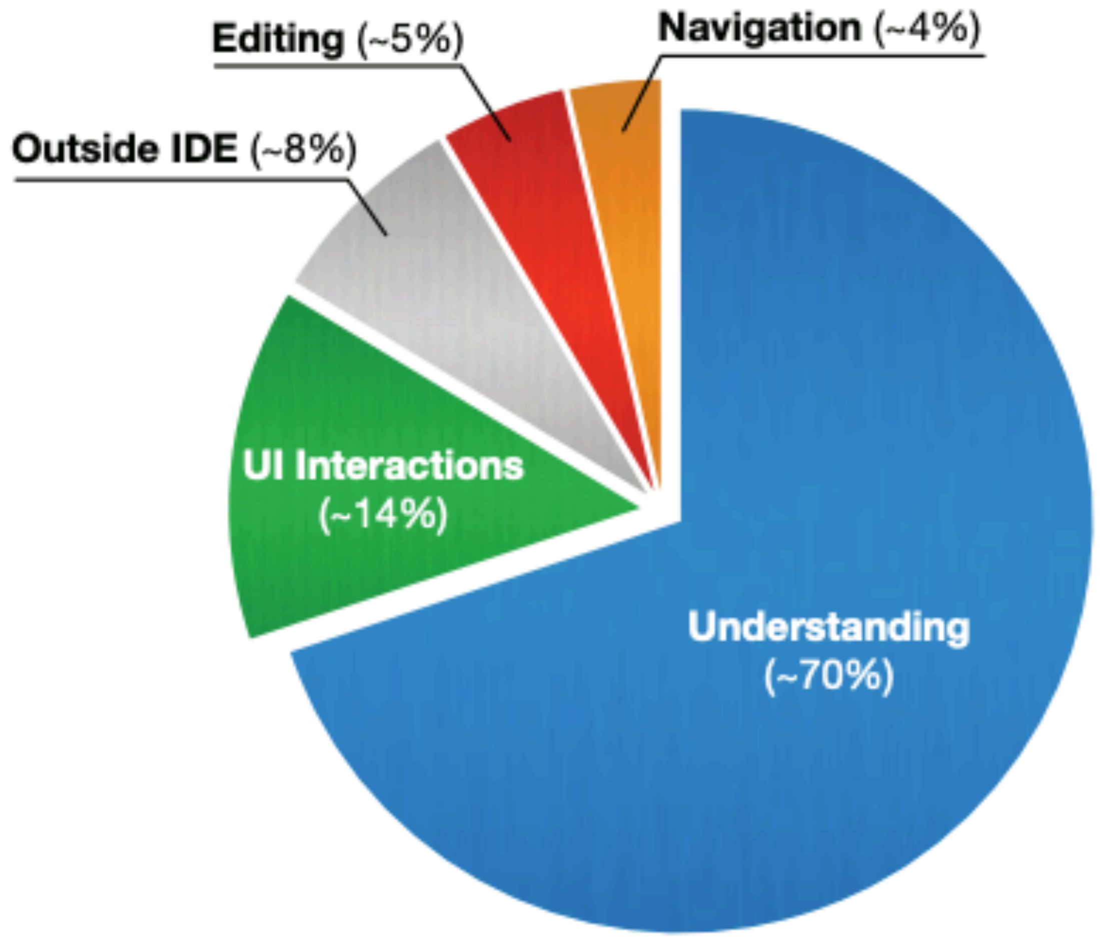
```
public class Person {
    private final String firstName;
    private final String lastName;
    private final Integer age;
    public Person(String firstName,
                  String lastName,
                  Integer age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
    public String getFirst() {
        return firstName;
    }
    public String getLast() {
        return lastName;
    }
    public Integer getAge() {
       return age;
    }
    public Boolean valid() {
       return age > 18;
    }
}
```

```
public static List<String> validByAge(List<Person> in) {
   List<Person> people = new ArrayList<Person>();
   for (Person p: in) {
     if (p.valid()) people.add(p);
   }

   Collections.sort(people, new Comparator<Person>() {
      public int compare(Person a, Person b) {
        return a.age() - b.age();
      }
   } );

   List<String> ret = new ArrayList<String>();
   for (Person p: people) {
     ret.add(p.first);
   }
   return ret;
}

List<Person> input = new ArrayList<Person>();
input.add(new Person("John", "Valid", 32));
input.add(new Person("John", "InValid", 17));
input.add(new Person("OtherJohn", "Valid", 19));

List<Person> output = validByAge(input)
```

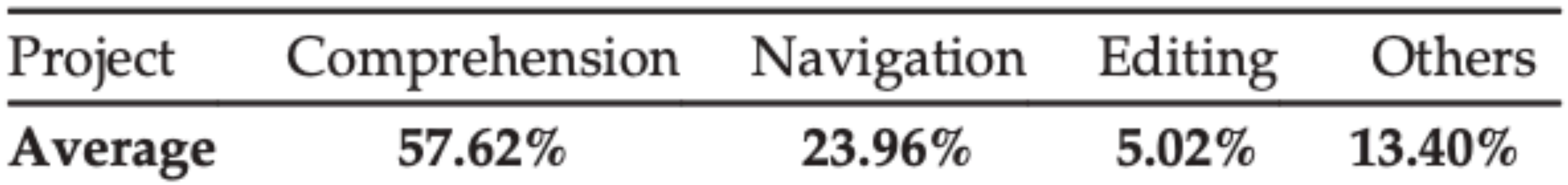
```
case class Person(val first: String, val last: String, val age: Int) {
  def valid: Boolean = age > 18
}

def validByAge(in: List[Person]) =
  in.filter(_.valid).sortBy(_.age).map(_.first)

validByAge(List(
  Person("John", "Valid", 32),
  Person("John", "Invalid", 17),
  Person("OtherJohn", "Valid", 19)))
```

David Pollak, "Beginning Scala"

Fact #1: Developers are reading, not writing



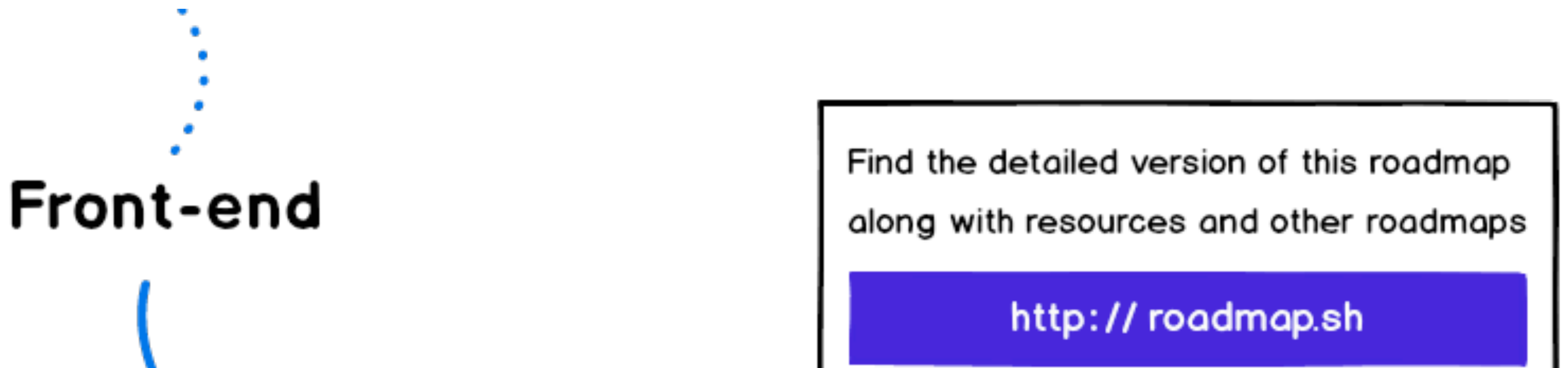
Minelli et al. "I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time" ICPC '15.



Project	Comprehension	Navigation	Editing	Others
Average	**57.62%**	**23.96%**	**5.02%**	**13.40%**

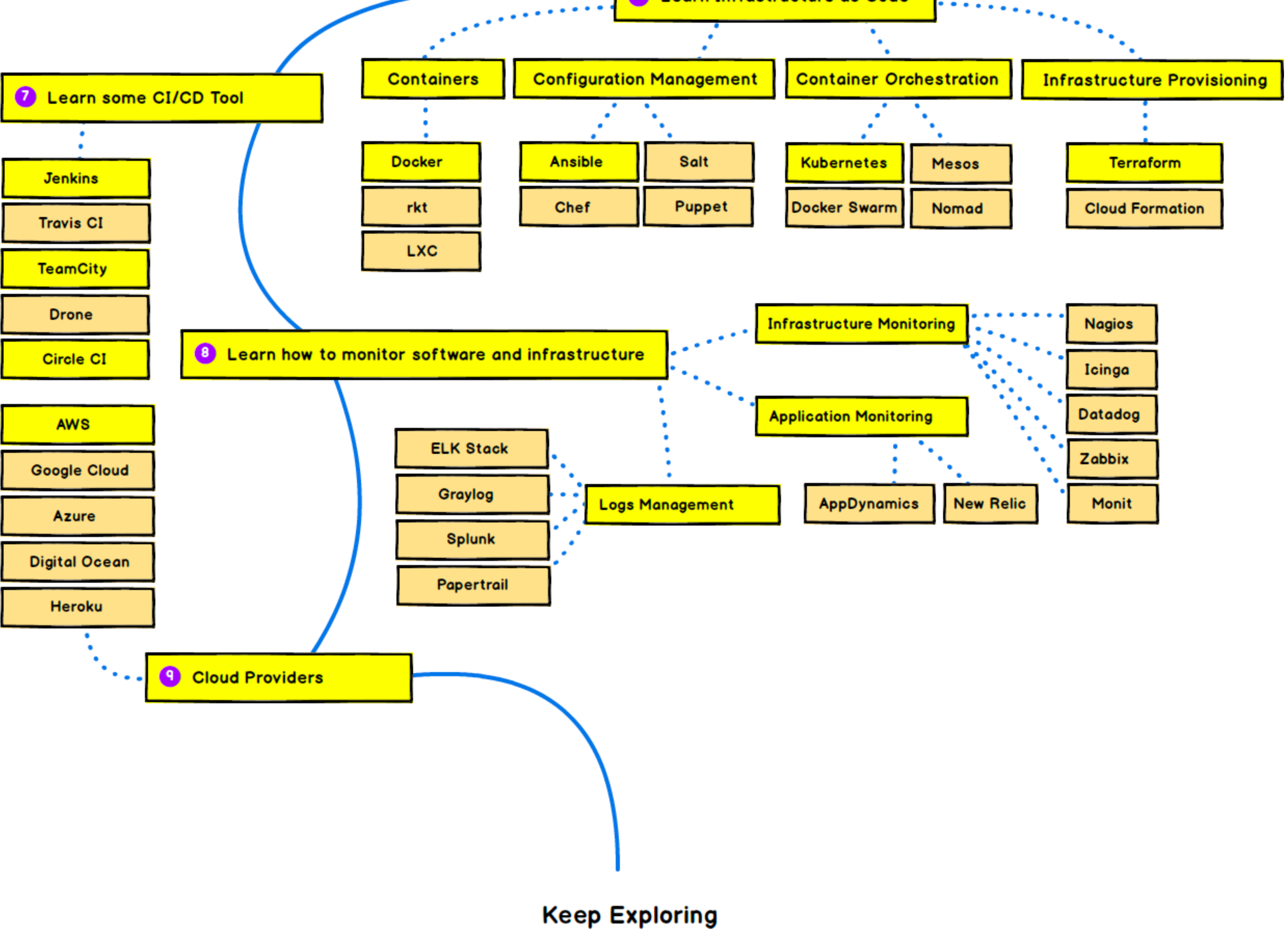
Xia et al. "Measuring Program Comprehension: A Large-Scale Field Study with Professionals." IEEE Trans. Softw. Eng, 2018.

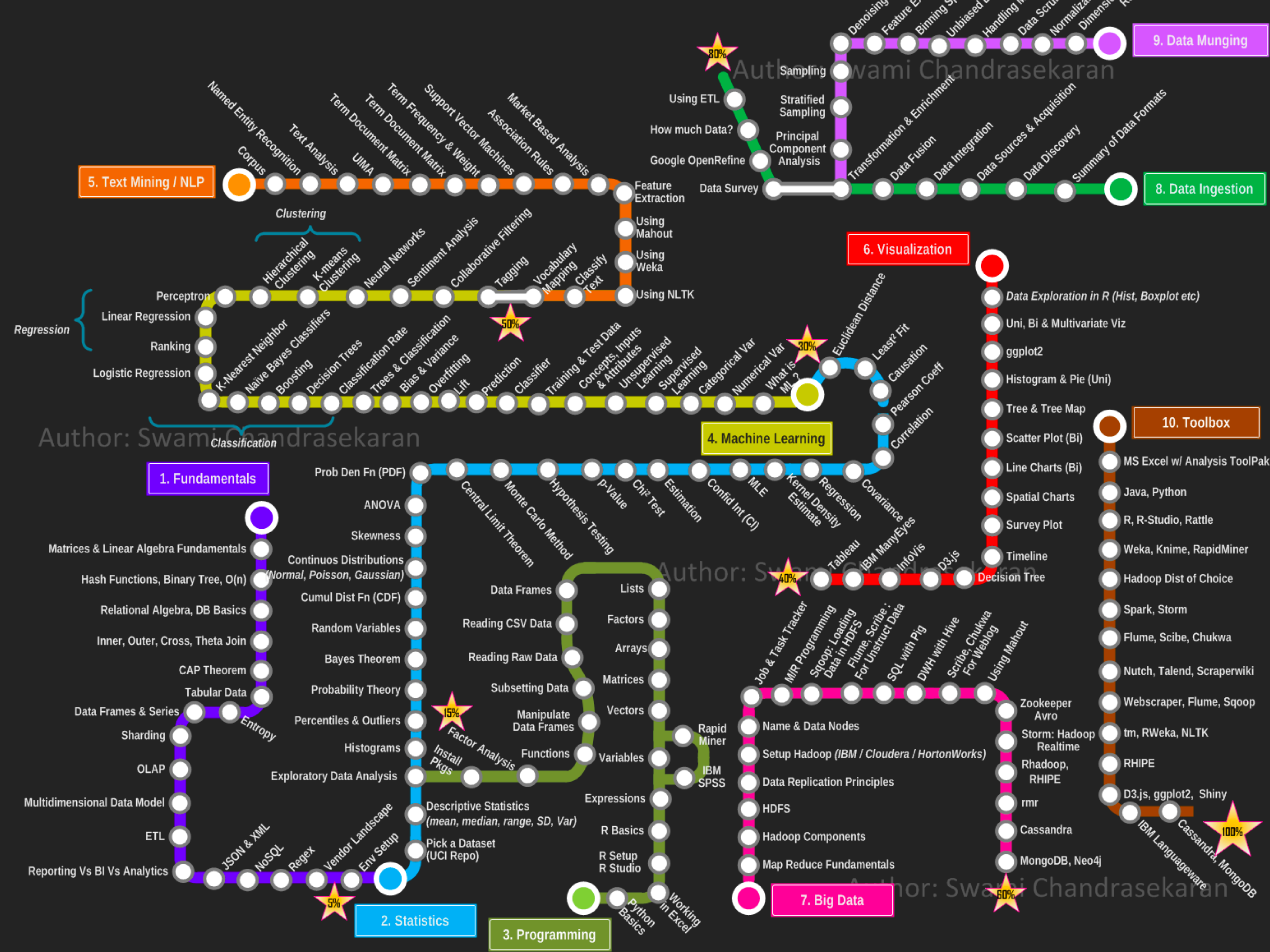
Fact #2: Software ecosystems are complex



Find the detailed version of this roadmap along with resources and other roadmaps

http:// roadmap.sh


Containers
Configuration Management
Container Orchestration
Infrastructure Provisioning
7 Learn some CI/CD Tool
Docker
rkt
LXC
Ansible
Salt
Chef
Puppet
Kubernetes
Mesos
Docker Swarm
Nomad
Terraform
Cloud Formation
Jenkins
Travis CI
TeamCity
Drone
Circle CI
Infrastructure Monitoring
Nagios
Icinga
Datadog
Zabbix
Monit
8 Learn how to monitor software and infrastructure
Application Monitoring
AppDynamics
New Relic
Logs Management
ELK Stack
Graylog
Splunk
Papertrail
AWS
Google Cloud
Azure
Digital Ocean
Heroku
9 Cloud Providers
Keep Exploring



1. Fundamentals
Matrices & Linear Algebra Fundamentals
Hash Functions, Binary Tree, O(n)
Relational Algebra, DB Basics
Inner, Outer, Cross, Theta Join
CAP Theorem
Tabular Data
Entropy
Data Frames & Series
Sharding
OLAP
Multidimensional Data Model
ETL
Reporting Vs BI Vs Analytics
JSON & XML
NoSQL
Regex
Vendor Landscape
Env Setup
5%
2. Statistics
Pick a Dataset (UCI Repo)
Descriptive Statistics (mean, median, range, SD, Var)
Exploratory Data Analysis
Histograms
Percentiles & Outliers
Probability Theory
Bayes Theorem
Random Variables
Cumul Dist Fn (CDF)
Continuos Distributions (Normal, Poisson, Gaussian)
Skewness
ANOVA
Prob Den Fn (PDF)
Central Limit Theorem
Monte Carlo Method
Hypothesis Testing
p-Value
Chi² Test
Estimation
Confid Int (CI)
MLE
Kernel Density Estimate
Regression
Covariance
Correlation
Pearson Coeff
Causation
Least² Fit
Euclidean Distance
15%
Factor Analysis
Install Pkgs
3. Programming
Python Basics
Working in Excel
R Setup R Studio
R Basics
Expressions
Variables
IBM SPSS
Rapid Miner
Vectors
Matrices
Arrays
Factors
Lists
Data Frames
Reading CSV Data
Reading Raw Data
Subsetting Data
Manipulate Data Frames
Functions
4. Machine Learning
What is ML ?
30%
Numerical Var
Categorical Var
Supervised Learning
Unsupervised Learning
Concepts, Inputs & Attributes
Training & Test Data
Classifier
Prediction
Lift
Overfitting
Bias & Variance
Trees & Classification
Classification Rate
Decision Trees
Boosting
Naive Bayes Classifiers
K-Nearest Neighbor
Classification
Logistic Regression
Ranking
Linear Regression
Regression
Perceptron
Hierarchical Clustering
K-means Clustering
Clustering
Neural Networks
Sentiment Analysis
Collaborative Filtering
Tagging
50%
5. Text Mining / NLP
Corpus
Named Entity Recognition
Text Analysis
UIMA
Term Document Matrix
Term Frequency & Weight
Support Vector Machines
Association Rules
Market Based Analysis
Feature Extraction
Using Mahout
Using Weka
Using NLTK
Classify Text
Vocabulary Mapping
6. Visualization
Data Exploration in R (Hist, Boxplot etc)
Uni, Bi & Multivariate Viz
ggplot2
Histogram & Pie (Uni)
Tree & Tree Map
Scatter Plot (Bi)
Line Charts (Bi)
Spatial Charts
Survey Plot
Timeline
Decision Tree
D3.js
InfoVis
IBM ManyEyes
Tableau
40%
7. Big Data
Map Reduce Fundamentals
Hadoop Components
HDFS
Data Replication Principles
Setup Hadoop (IBM / Cloudera / HortonWorks)
Name & Data Nodes
Job & Task Tracker
MR Programming
Sqoop: Loading Data in HDFS
Flume, Scribe: For Unstruct Data
SQL with Pig
DWH with Hive
Scribe, Chukwa For Weblog
Using Mahout
Zookeeper Avro
Storm: Hadoop Realtime
Rhadoop, RHIPE
rmr
Cassandra
MongoDB, Neo4j
60%
8. Data Ingestion
Summary of Data Formats
Data Discovery
Data Sources & Acquisition
Data Integration
Data Fusion
Transformation & Enrichment
Data Survey
Google OpenRefine
How much Data?
Using ETL
80%
9. Data Munging
Dimensio
Normaliza
Data Scru
Handling
Unbiased
Binning S
Feature E
Denoising
R
Sampling
Stratified Sampling
Principal Component Analysis
10. Toolbox
MS Excel w/ Analysis ToolPak
Java, Python
R, R-Studio, Rattle
Weka, Knime, RapidMiner
Hadoop Dist of Choice
Spark, Storm
Flume, Scibe, Chukwa
Nutch, Talend, Scraperwiki
Webscraper, Flume, Sqoop
tm, RWeka, NLTK
RHIPE
D3.js, ggplot2, Shiny
IBM Languageware
Cassandra, MongoDB
100%
Author: Swami Chandrasekaran

td::move_if_noexcept

Defined in header <utility>

```
template< class T >
typename std::conditional<
    !std::is_nothrow_move_constructible<T>::value && std::is_copy_constructible<T>::value,
    const T&,
    T&&
>::type move_if_noexcept(T& x) noexcept;
```

(sinc
C++
(until
C++

```
template< class T >
constexpr typename std::conditional<
    !std::is_nothrow_move_constructible<T>::value && std::is_copy_constructible<T>::value,
    const T&,
    T&&
>::type move_if_noexcept(T& x) noexcept;
```

(sinc
C++

ove_if_noexcept obtains an rvalue reference to its argument if its move constructor does not throw exceptions or if
here is no copy constructor (move-only type), otherwise obtains an lvalue reference to its argument. It is typically
sed to combine move semantics with strong exception guarantee.

Parameters

 - the object to be moved or copied

Return value

`std::move(x)` or `x`, depending on exception guarantees.

Notes

his is used, for example, by `std::vector::resize`, which may have to allocate new storage and then move or copy
lements from old storage to new storage. If an exception occurs during this operation, `std::vector::resize` undoes
verything it did to this point, which is only possible if `std::move_if_noexcept` was used to decide whether to use
ove construction or copy construction. (unless copy constructor is not available, in which case move constructor is
sed either way and the strong exception guarantee may be waived)

Example

6.7.3.1 Formal definition of `restrict`

1 Let **`D`** be a declaration of an ordinary identifier that provides a means of designating an object **`P`** as a restrict-qualified pointer to type **`T`**.

2 If **`D`** appears inside a block and does not have storage class **`extern`**, let **`B`** denote the block. If **`D`** appears in the list of parameter declarations of a function definition, let **`B`** denote the associated block. Otherwise, let **`B`** denote the block of **`main`** (or the block of whatever function is called at program startup in a freestanding environment).

3 In what follows, a pointer expression **`E`** is said to be *based* on object **`P`** if (at some sequence point in the execution of **`B`** prior to the evaluation of **`E`**) modifying **`P`** to point to a copy of the array object into which it formerly pointed would change the value of **`E`**.119) Note that "based" is defined only for expressions with pointer types.

4 During each execution of **`B`**, let **`L`** be any lvalue that has **`&L`** based on **`P`**. If **`L`** is used to access the value of the object **`X`** that it designates, and **`X`** is also modified (by any means), then the following requirements apply: **`T`** shall not be const-qualified. Every other lvalue used to access the value of **`X`** shall also have its address based on **`P`**. Every access that modifies **`X`** shall be considered also to modify **`P`**, for the purposes of this subclause. If **`P`** is assigned the value of a pointer expression **`E`** that is based on another restricted pointer object **`P2`**, associated with block **`B2`**, then either the execution of **`B2`** shall begin before the execution of **`B`**, or the execution of **`B2`** shall end prior to the assignment. If these requirements are not met, then the behavior is undefined.

5 Here an execution of **`B`** means that portion of the execution of the program that would correspond to the lifetime of an object with scalar type and automatic storage duration

---

119) In other words, **`E`** depends on the value of **`P`** itself rather than on the value of an object referenced

Fact #3: Dennard scaling is dead

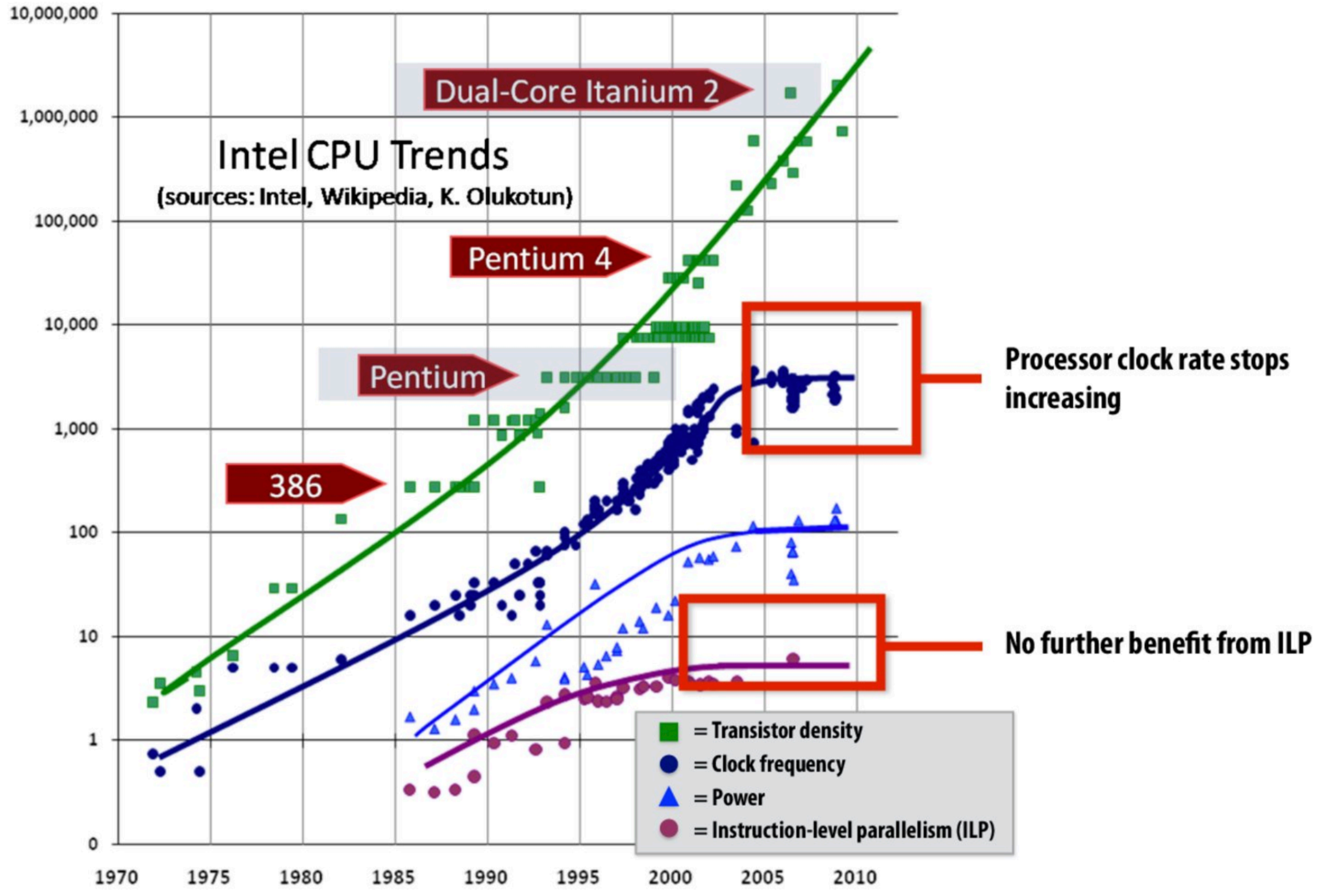
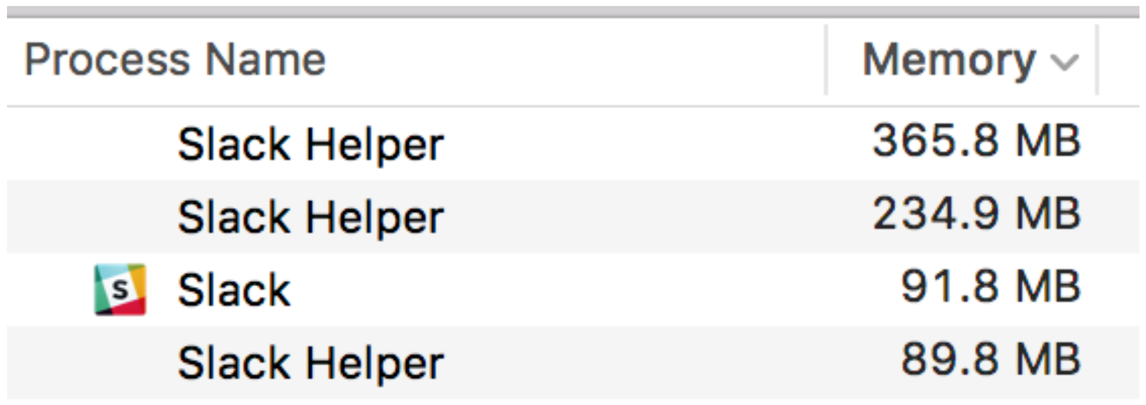


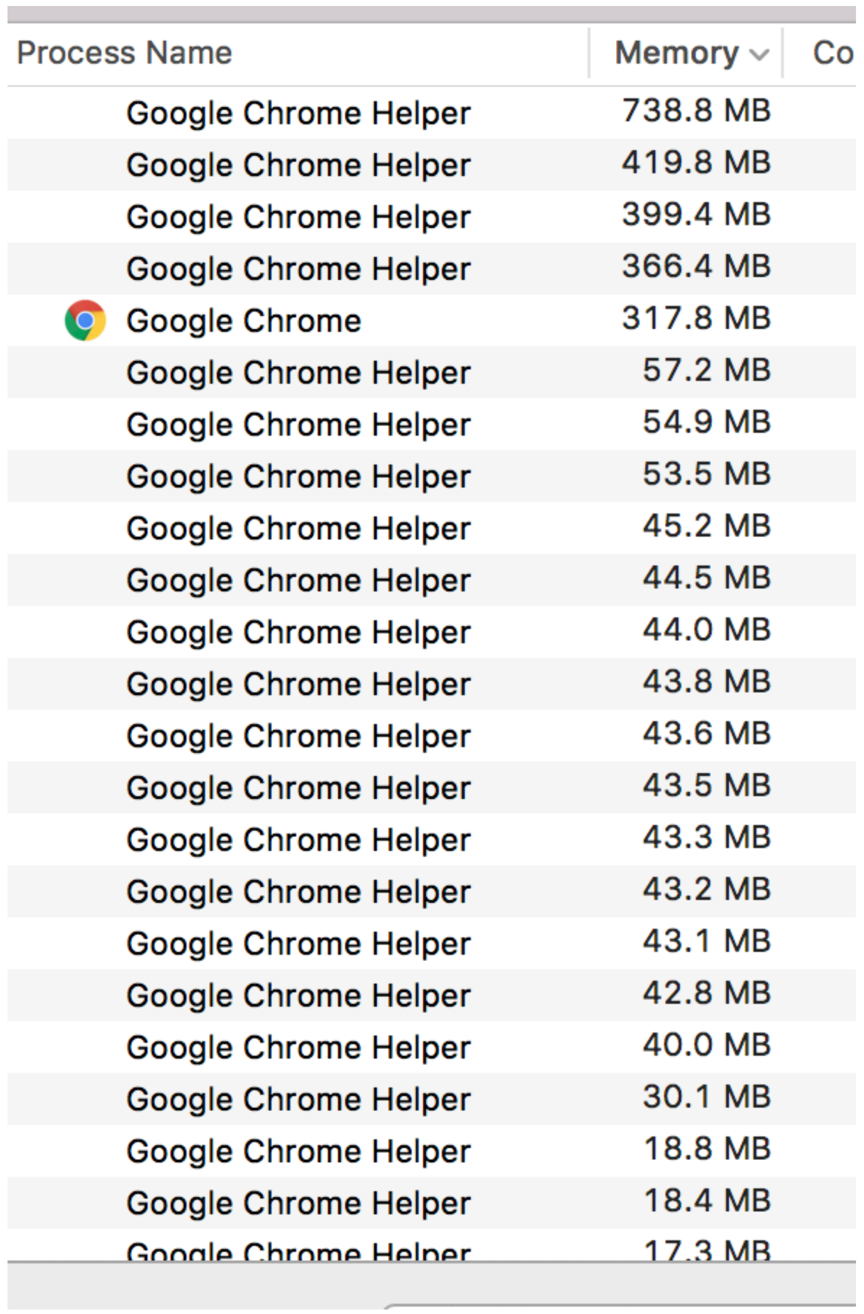
Image credit: "The free Lunch is Over" by Herb Sutter, Dr. Dobbs 2005

Memory usage continues to scale



Process Name	Memory
Slack Helper	365.8 MB
Slack Helper	234.9 MB
Slack	91.8 MB
Slack Helper	89.8 MB

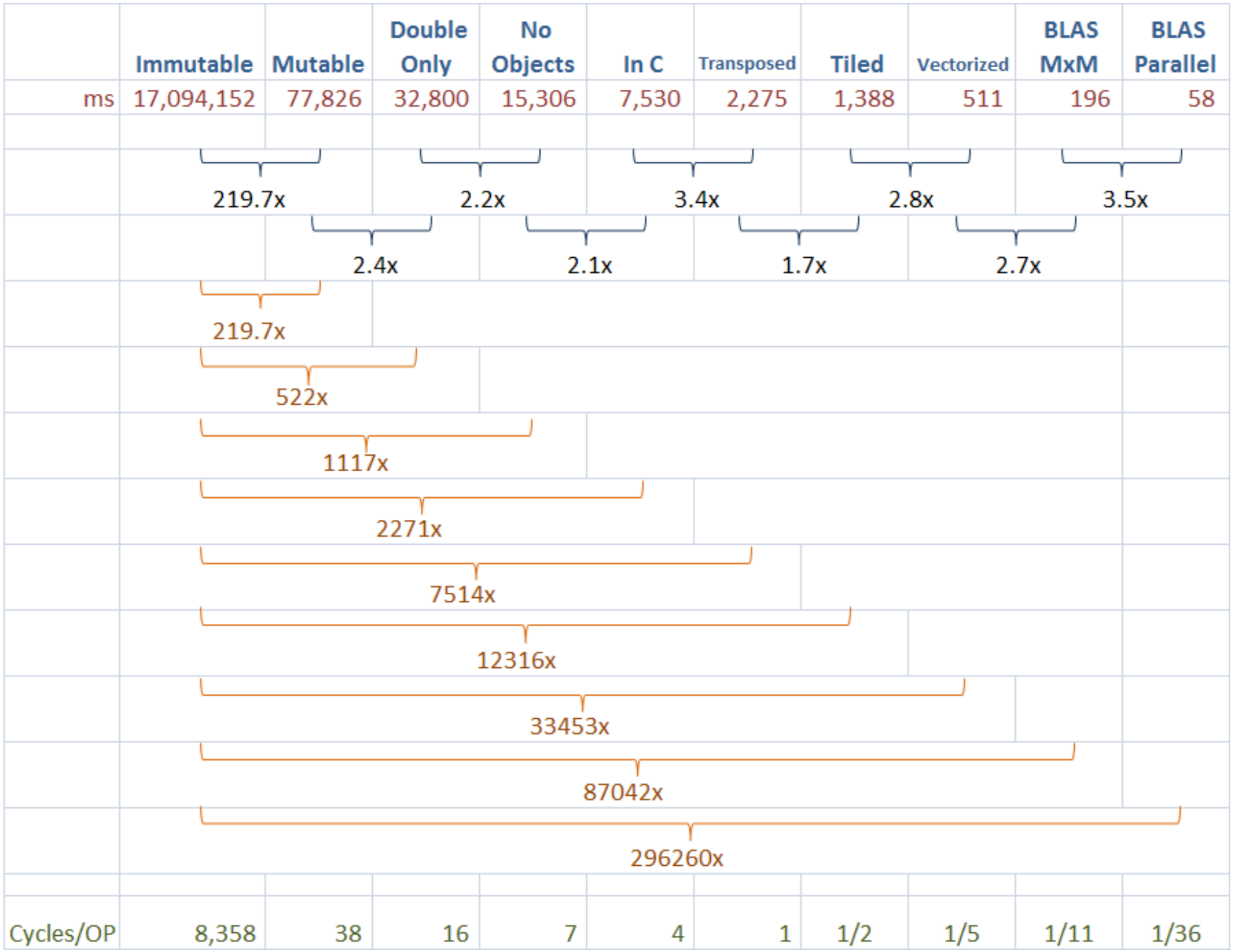
700 MB!



Process Name	Memory	Co
Google Chrome Helper	738.8 MB	
Google Chrome Helper	419.8 MB	
Google Chrome Helper	399.4 MB	
Google Chrome Helper	366.4 MB	
Google Chrome	317.8 MB	
Google Chrome Helper	57.2 MB	
Google Chrome Helper	54.9 MB	
Google Chrome Helper	53.5 MB	
Google Chrome Helper	45.2 MB	
Google Chrome Helper	44.5 MB	
Google Chrome Helper	44.0 MB	
Google Chrome Helper	43.8 MB	
Google Chrome Helper	43.6 MB	
Google Chrome Helper	43.5 MB	
Google Chrome Helper	43.3 MB	
Google Chrome Helper	43.2 MB	
Google Chrome Helper	43.1 MB	
Google Chrome Helper	42.8 MB	
Google Chrome Helper	40.0 MB	
Google Chrome Helper	30.1 MB	
Google Chrome Helper	18.8 MB	
Google Chrome Helper	18.4 MB	
Google Chrome Helper	17.3 MB	

3 GB!!!

Huge perf gap for most programs



	Immutable	Mutable	Double Only	No Objects	In C	Transposed	Tiled	Vectorized	BLAS MxM	BLAS Parallel
ms	17,094,152	77,826	32,800	15,306	7,530	2,275	1,388	511	196	58
Cycles/OP	8,358	38	16	7	4	1	1/2	1/5	1/11	1/36

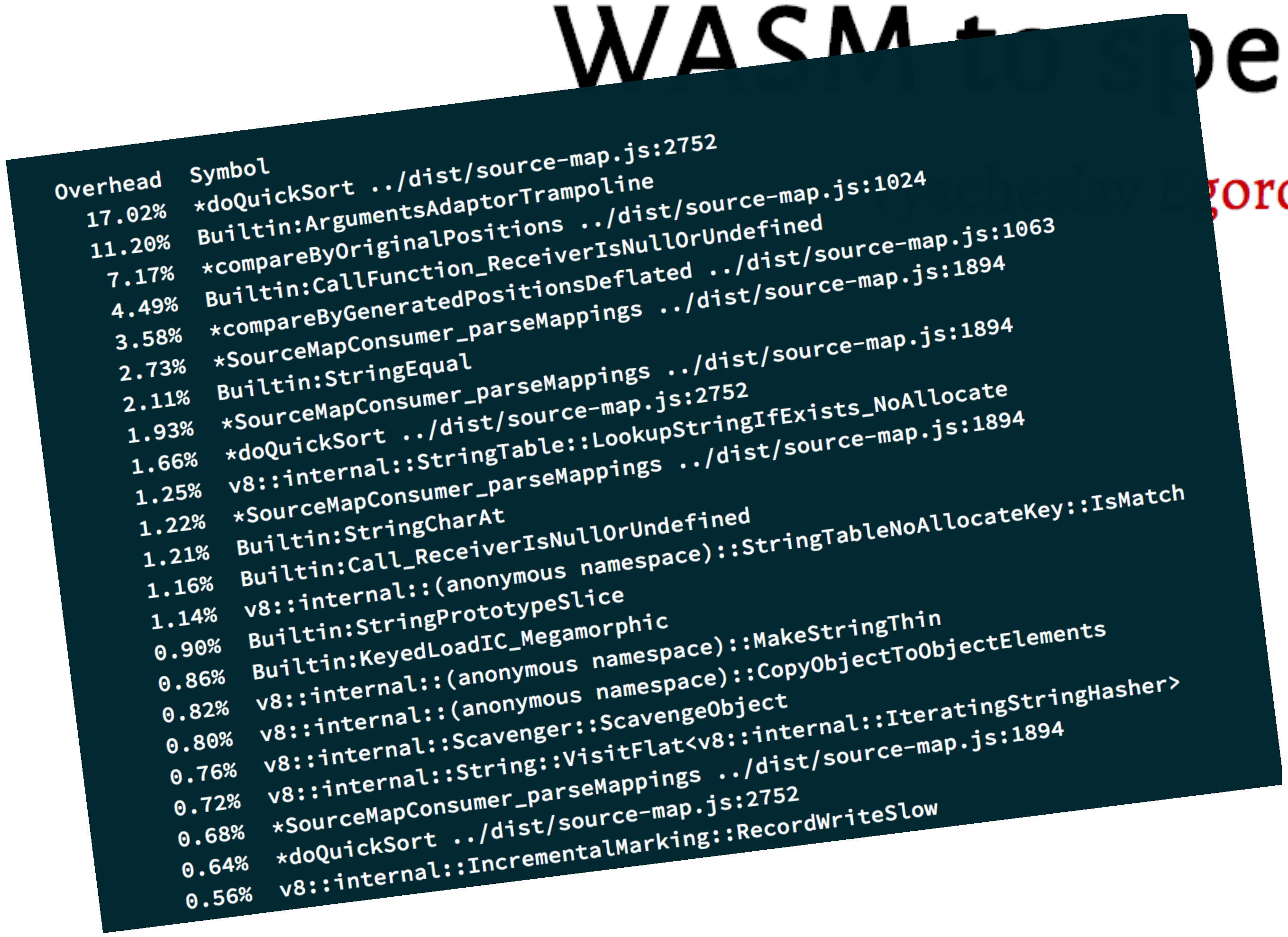
Saman Amarasinghe, MIT 6.172 "Performance Engineering". 2009

Maybe you don't need Rust and WASM to speed up your JS

Vyacheslav Egorov on 03 feb 2018

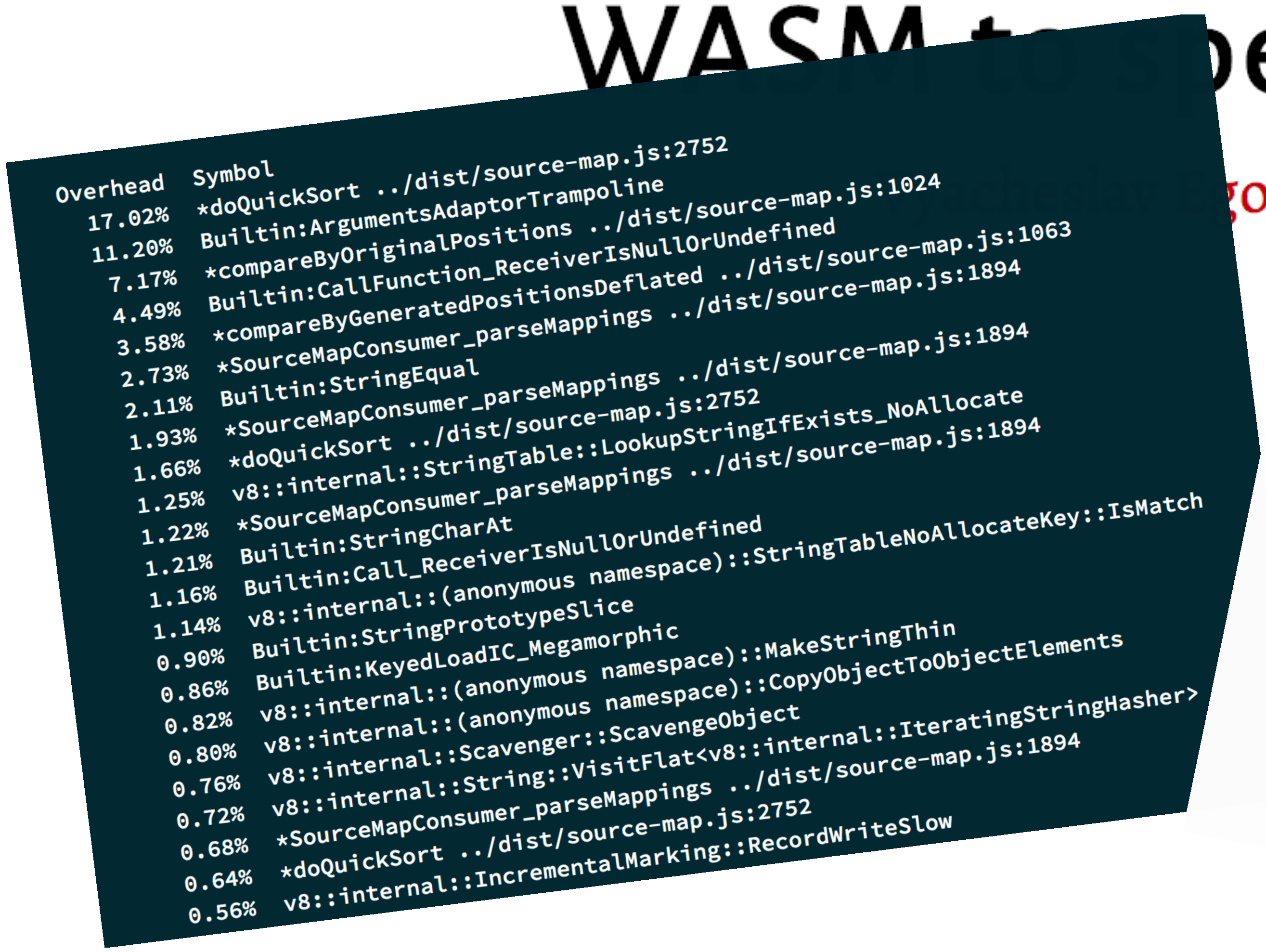
Maybe you don't need Rust and WASM to speed up your JS

Egorov on 03 feb 2018

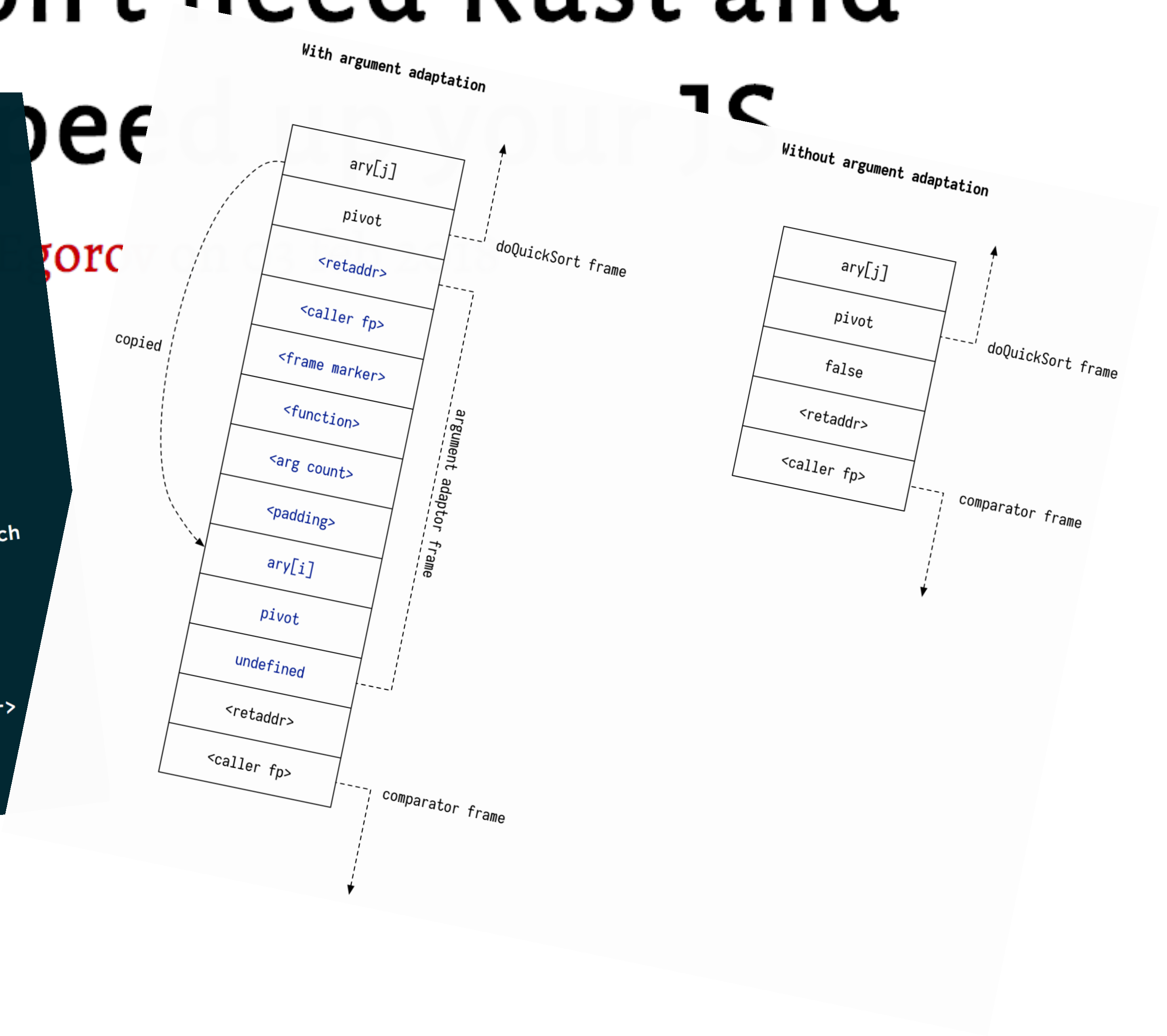


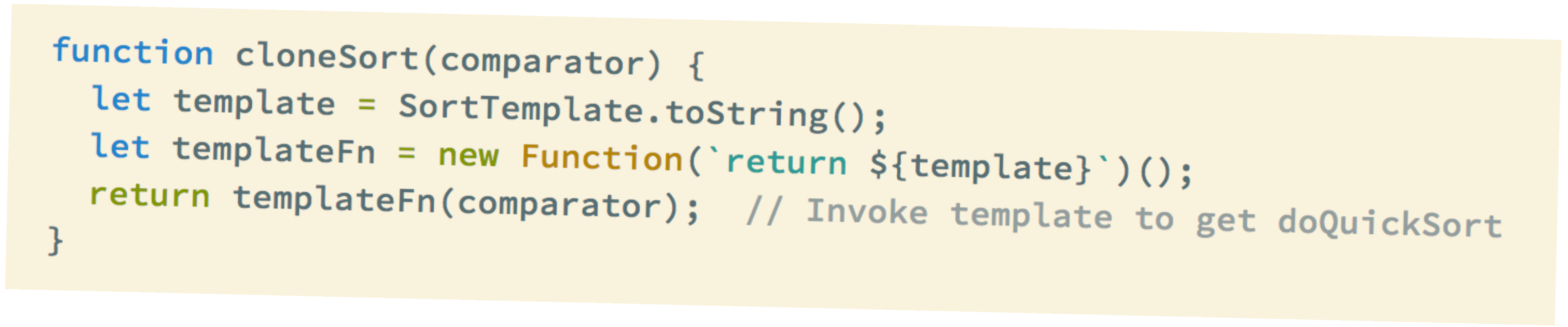
```
Overhead  Symbol
  17.02%  *doQuickSort ../dist/source-map.js:2752
  11.20%  Builtin:ArgumentsAdaptorTrampoline
   7.17%  *compareByOriginalPositions ../dist/source-map.js:1024
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```

Maybe you don't need Rust and WASM to speed up your JS



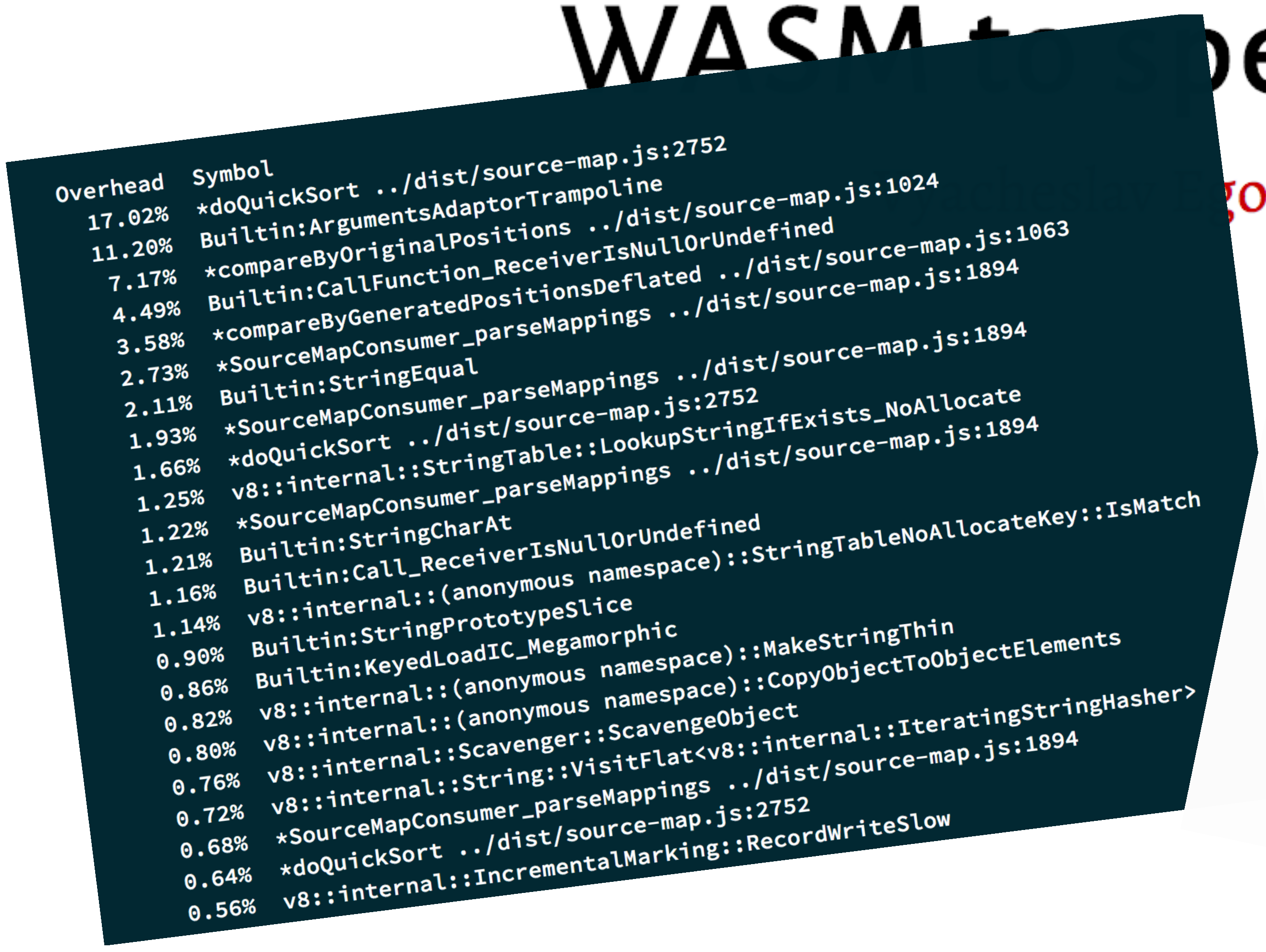
```
Overhead  Symbol
  17.02%  *doQuickSort ../dist/source-map.js:2752
  11.20%  Builtin:ArgumentsAdaptorTrampoline
   7.17%  *compareByOriginalPositions ../dist/source-map.js:1024
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```



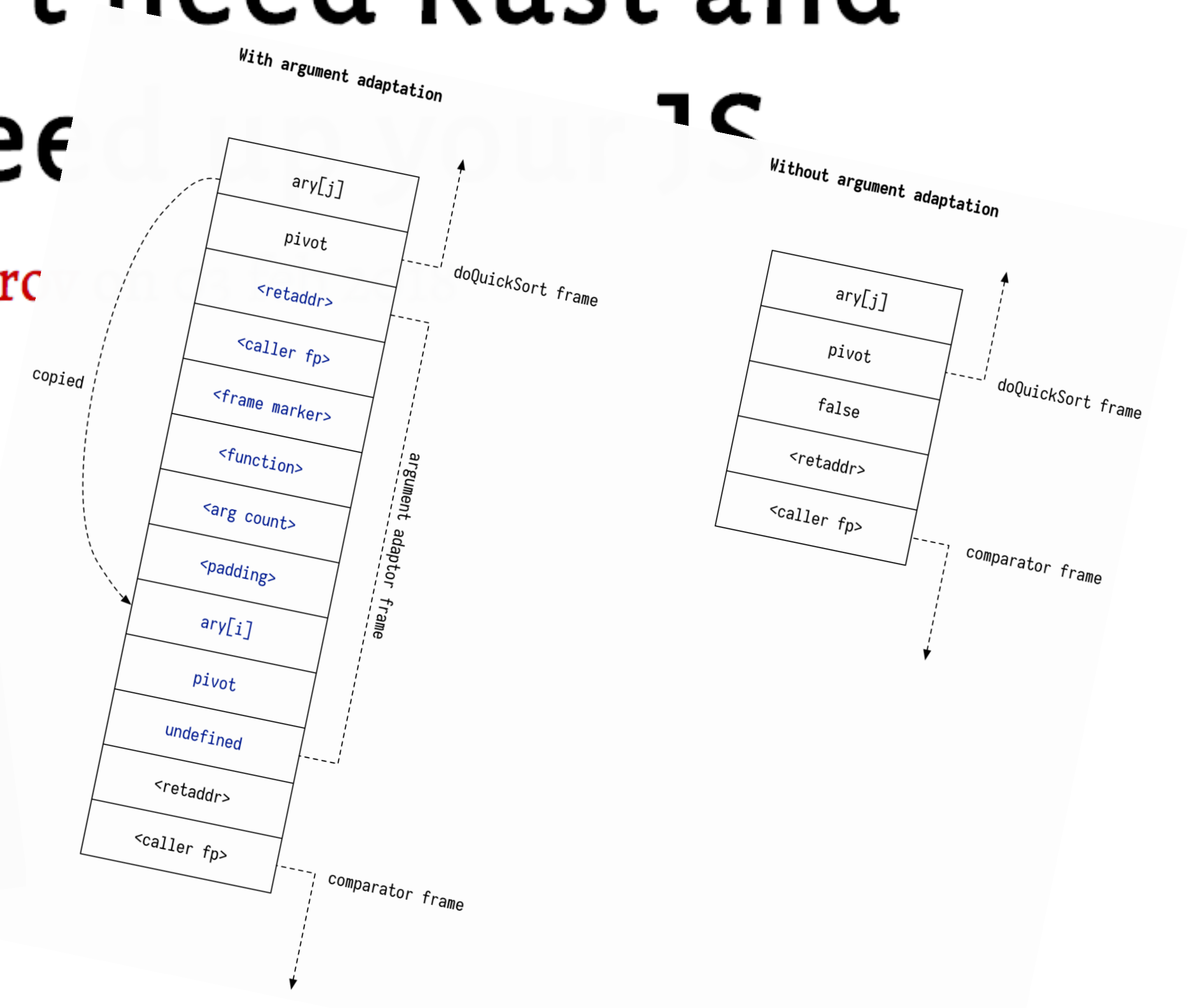


```
function cloneSort(comparator) {
  let template = SortTemplate.toString();
  let templateFn = new Function(`return ${template}`)();
  return templateFn(comparator);  // Invoke template to get doQuickSort
}
```

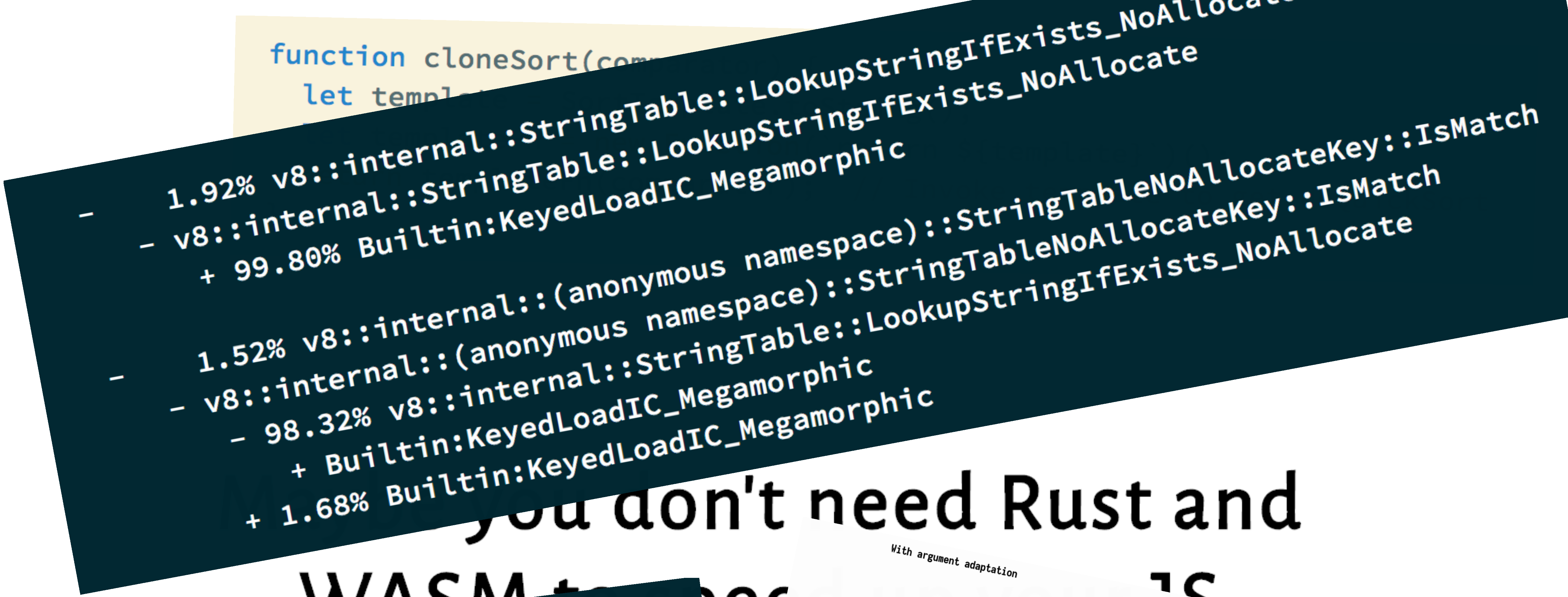
Maybe you don't need Rust and WASM to speed up your JS



```
Overhead  Symbol
  17.02%  *doQuickSort ../dist/source-map.js:2752
  11.20%  Builtin:ArgumentsAdaptorTrampoline
   7.17%  *compareByOriginalPositions ../dist/source-map.js:1024
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```



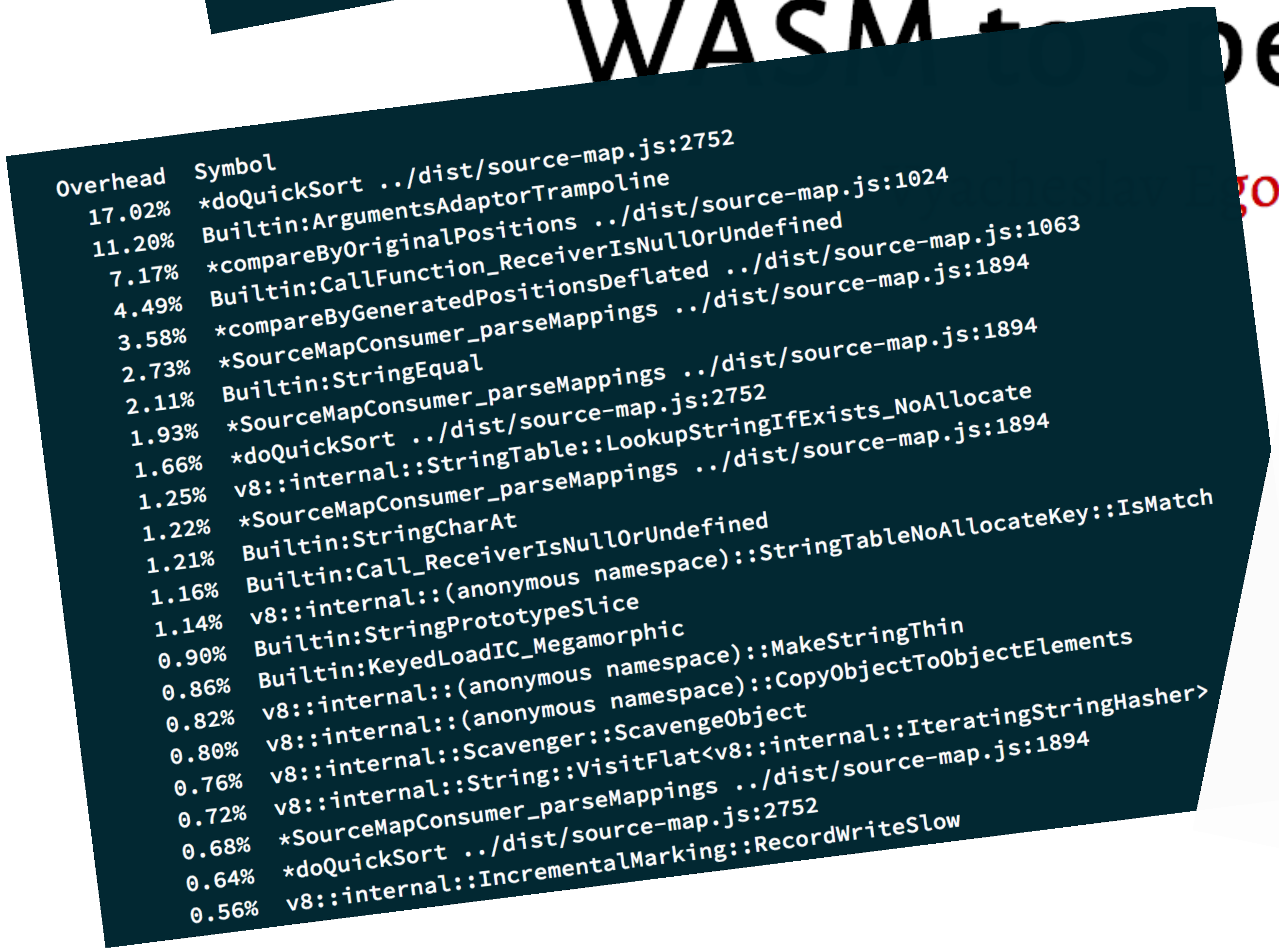
```
function cloneSort(com
  let templ
```



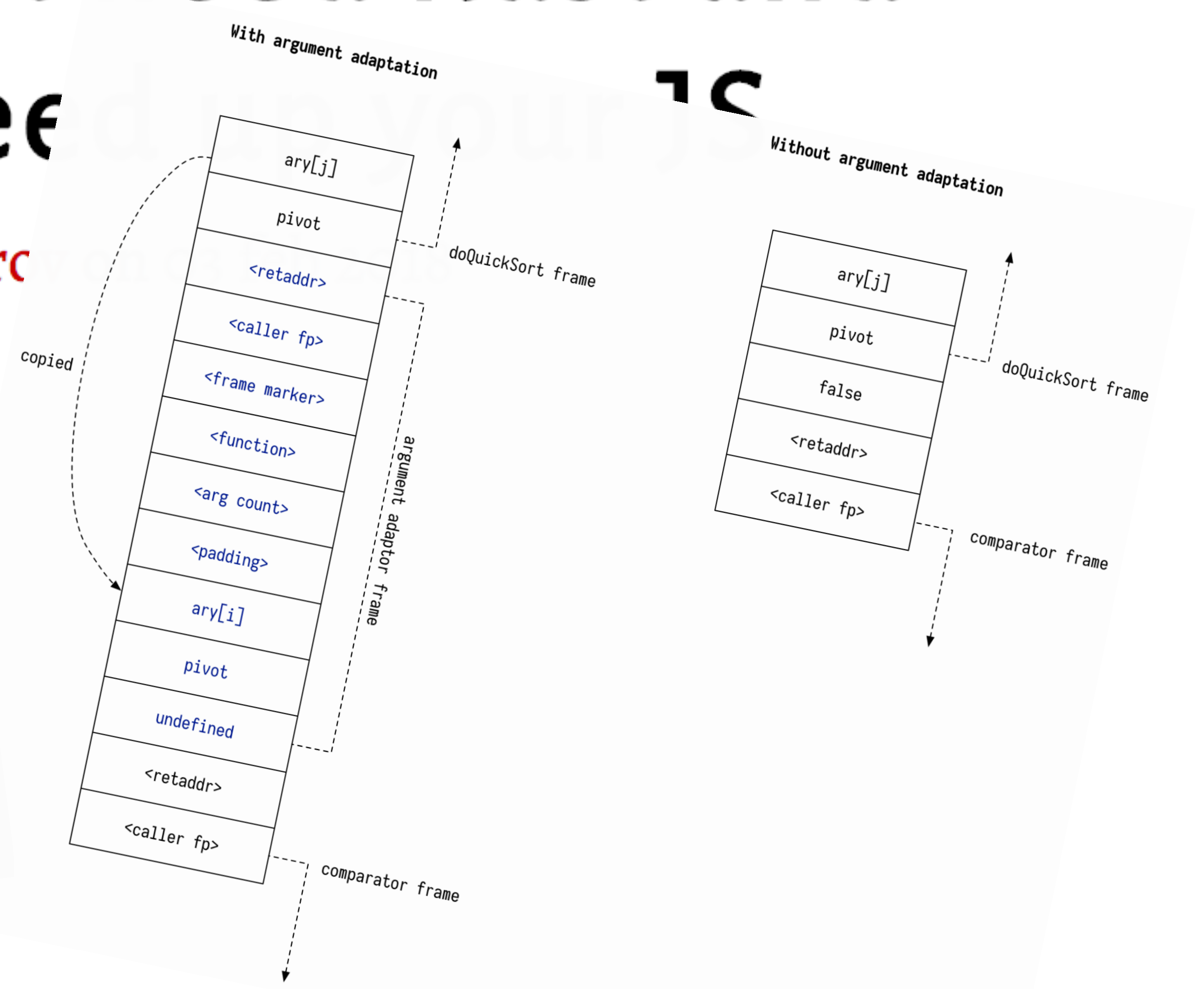
```
-     1.92% v8::internal::StringTable::LookupStringIfExists_NoAlloca
  - v8::internal::StringTable::LookupStringIfExists_NoAllocate
    + 99.80% Builtin:KeyedLoadIC_Megamorphic

-     1.52% v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
  - v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
    - 98.32% v8::internal::StringTable::LookupStringIfExists_NoAllocate
      + Builtin:KeyedLoadIC_Megamorphic
  + 1.68% Builtin:KeyedLoadIC_Megamorphic
```

Maybe you don't need Rust and WASM to speed up your JS

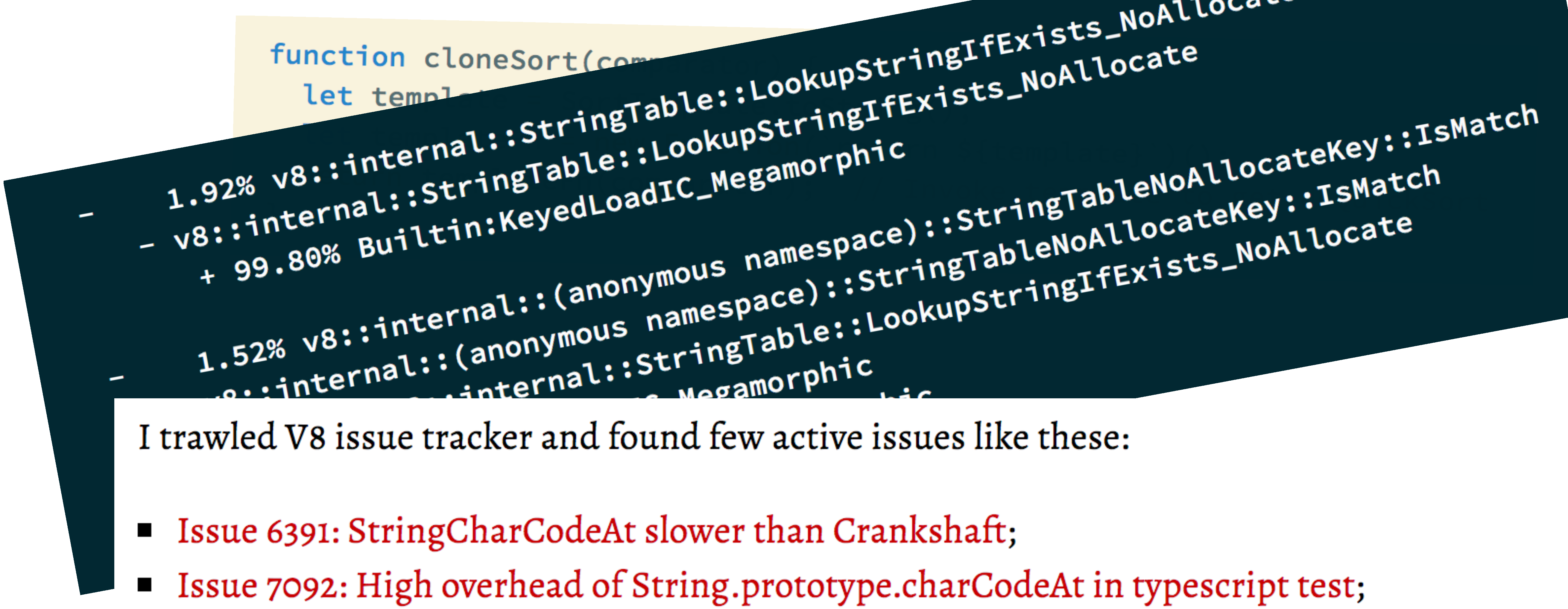
Egorov



```
Overhead  Symbol
  17.02%  *doQuickSort ../dist/source-map.js:2752
  11.20%  Builtin:ArgumentsAdaptorTrampoline
   7.17%  *compareByOriginalPositions ../dist/source-map.js:1024
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```



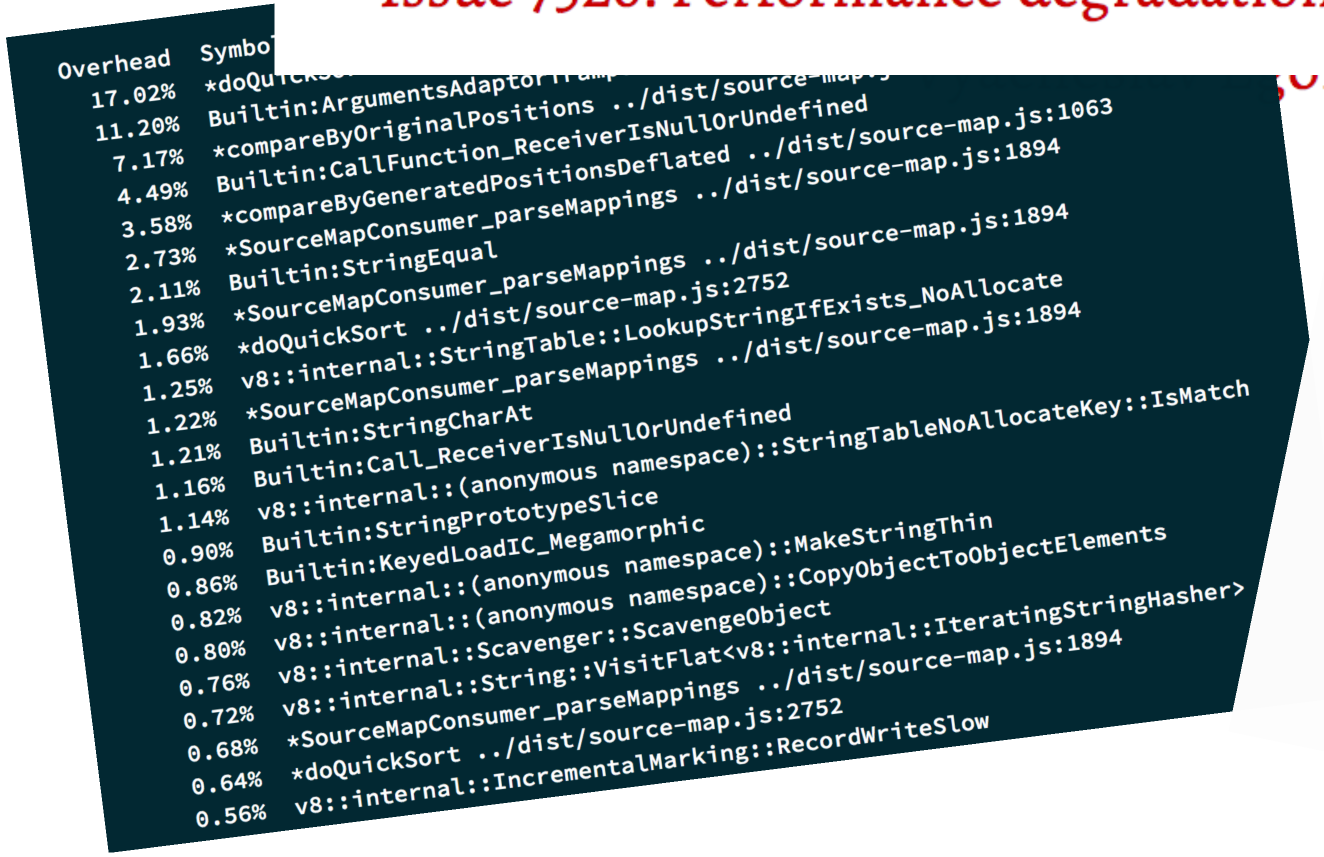
```
function cloneSort(com
  let templ
```



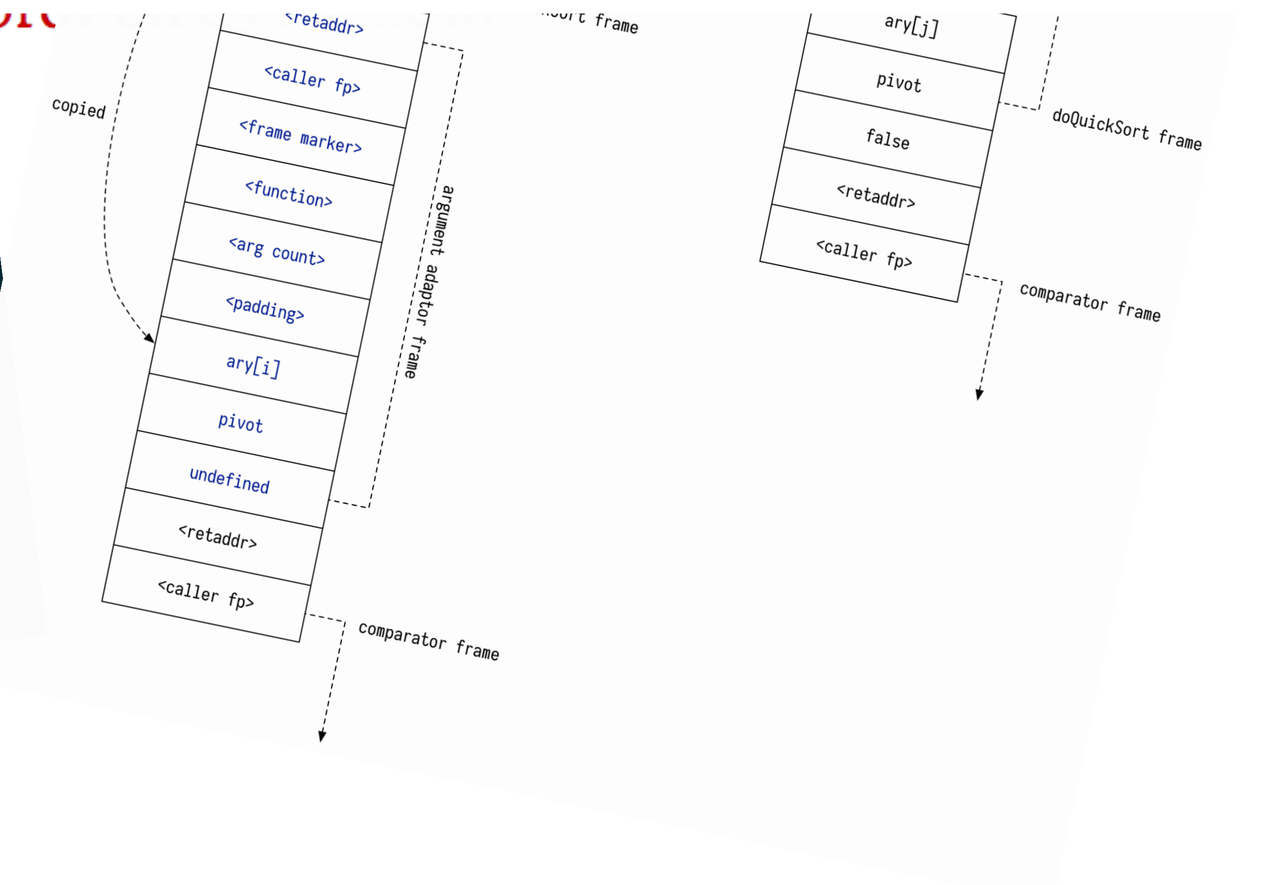
```
-    1.92% v8::internal::StringTable::LookupStringIfExists_NoAlloca
   - v8::internal::StringTable::LookupStringIfExists_NoAllocate
      + 99.80% Builtin:KeyedLoadIC_Megamorphic

-    1.52% v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   - v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
      - v8::internal::StringTable::LookupStringIfExists_NoAllocate
           Megamorphic
```

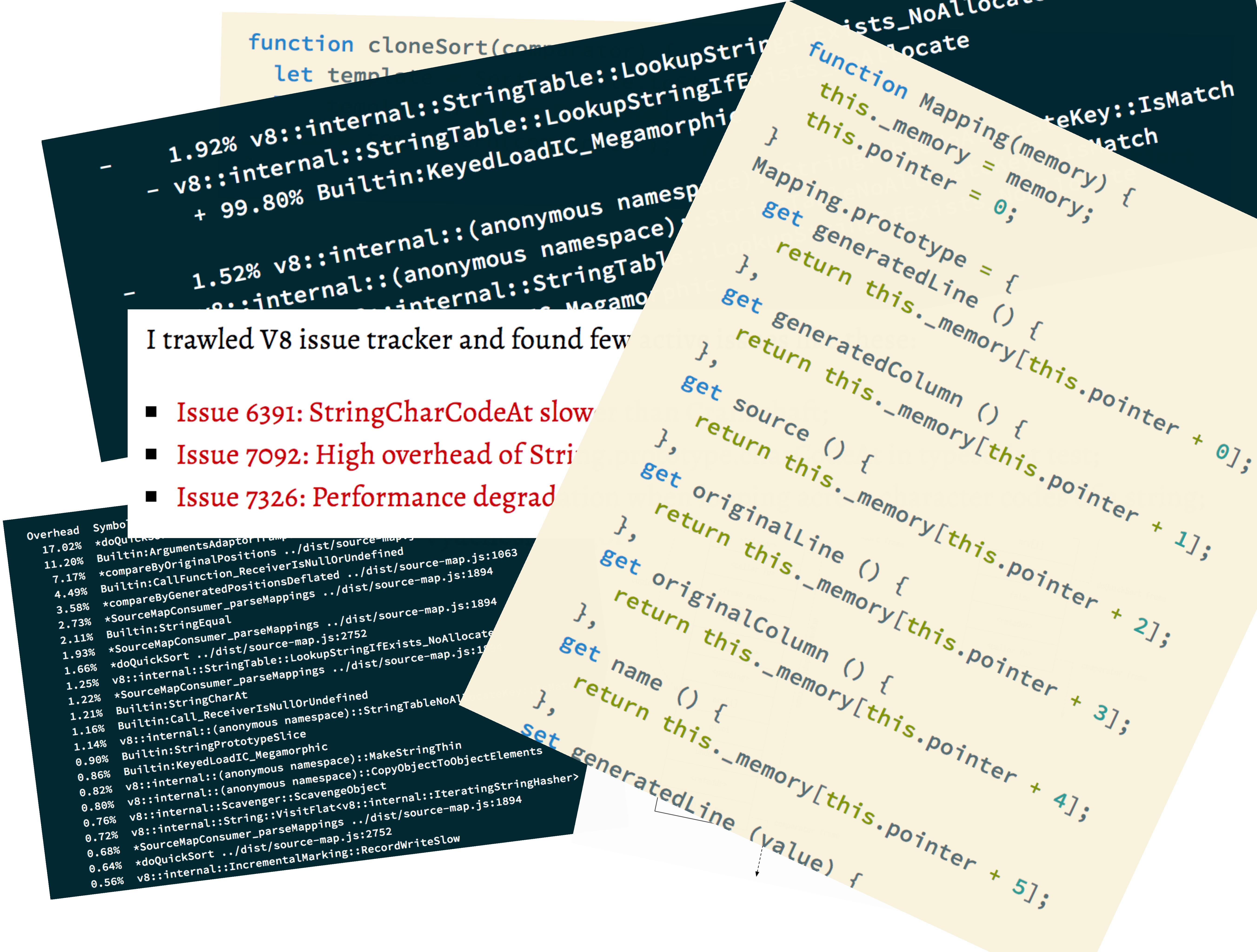
I trawled V8 issue tracker and found few active issues like these:

- Issue 6391: StringCharCodeAt slower than Crankshaft;
- Issue 7092: High overhead of String.prototype.charCodeAt in typescript test;
- Issue 7326: Performance degradation when looping across character codes of a string;



```
Overhead  Symbo
  17.02%  *doQu
  11.20%  Builtin:ArgumentsAdaptorTramp
   7.17%  *compareByOriginalPositions ../dist/source-map.j
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```





```
function cloneSort(com
  let templ
```

```
-     1.92% v8::internal::StringTable::LookupStrin
   - v8::internal::StringTable::LookupStringIfE
      + 99.80% Builtin:KeyedLoadIC_Megamorphi

-     1.52% v8::internal::(anonymous namesp
   v8::internal::(anonymous namespace)
        v8::internal::StringTabl
```

```
ists_NoAlloca
llocate
eKey::IsMatch
Match
```

I trawled V8 issue tracker and found few

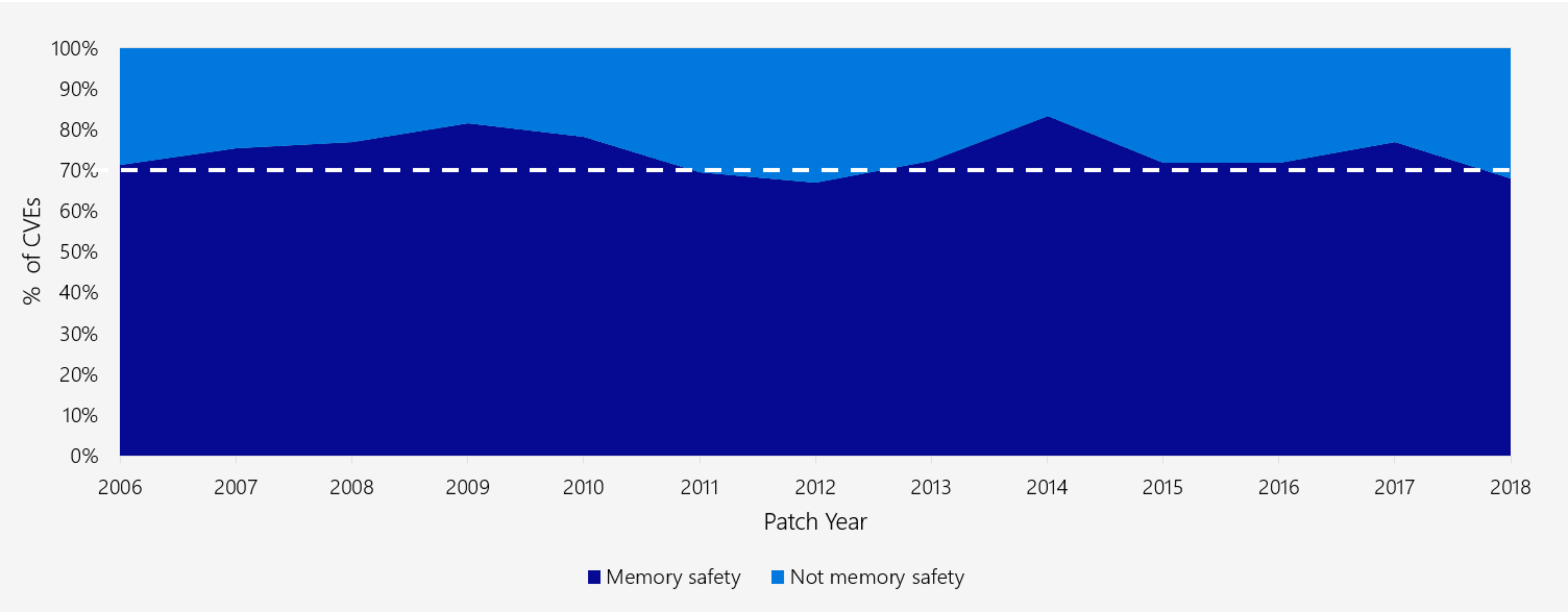
- Issue 6391: StringCharCodeAt slow
- Issue 7092: High overhead of Stri
- Issue 7326: Performance degrad

```
Overhead  Symbo
  17.02%  *doQuickSo
  11.20%  Builtin:ArgumentsAdaptorTramp              ../dist/source-map.j
   7.17%  *compareByOriginalPositions ../dist/source-map.j
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAl
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```

```
function Mapping(memory) {
  this._memory = memory;
  this.pointer = 0;
}
Mapping.prototype = {
  get generatedLine () {
    return this._memory[this.pointer + 0];
  },
  get generatedColumn () {
    return this._memory[this.pointer + 1];
  },
  get source () {
    return this._memory[this.pointer + 2];
  },
  get originalLine () {
    return this._memory[this.pointer + 3];
  },
  get originalColumn () {
    return this._memory[this.pointer + 4];
  },
  get name () {
    return this._memory[this.pointer + 5];
  },
  set generatedLine (value) {
```

Fact #4: Bugs are not improving

"The majority of vulnerabilities fixed and with a CVE assigned are caused by developers inadvertently inserting memory corruption bugs into their C and C++ code."



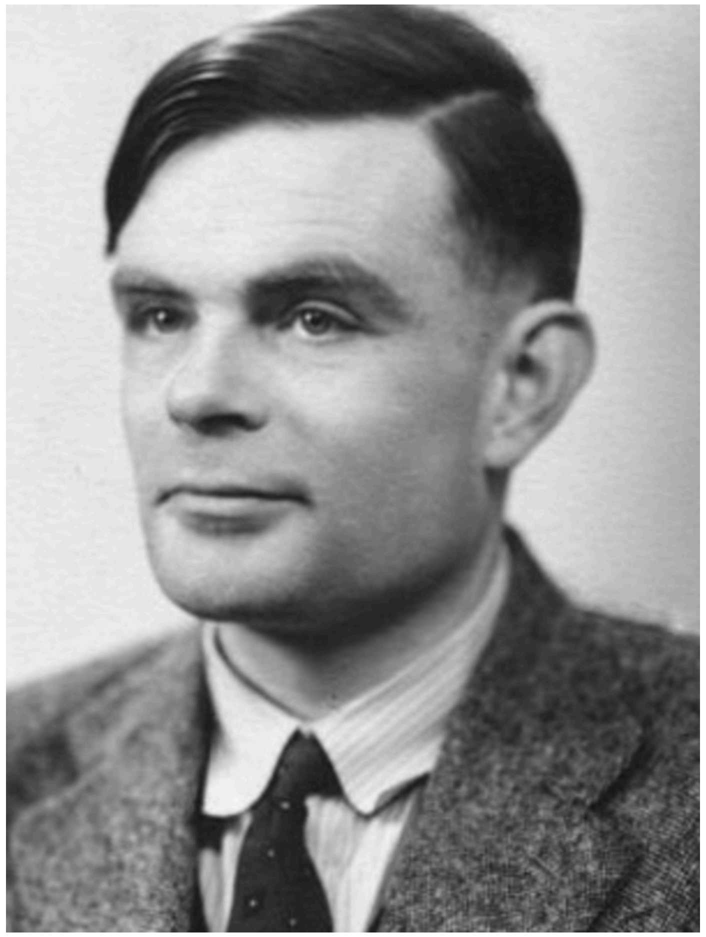
Microsoft Security Response Center. "A proactive approach to more secure code." 2019

Facts of modern software development

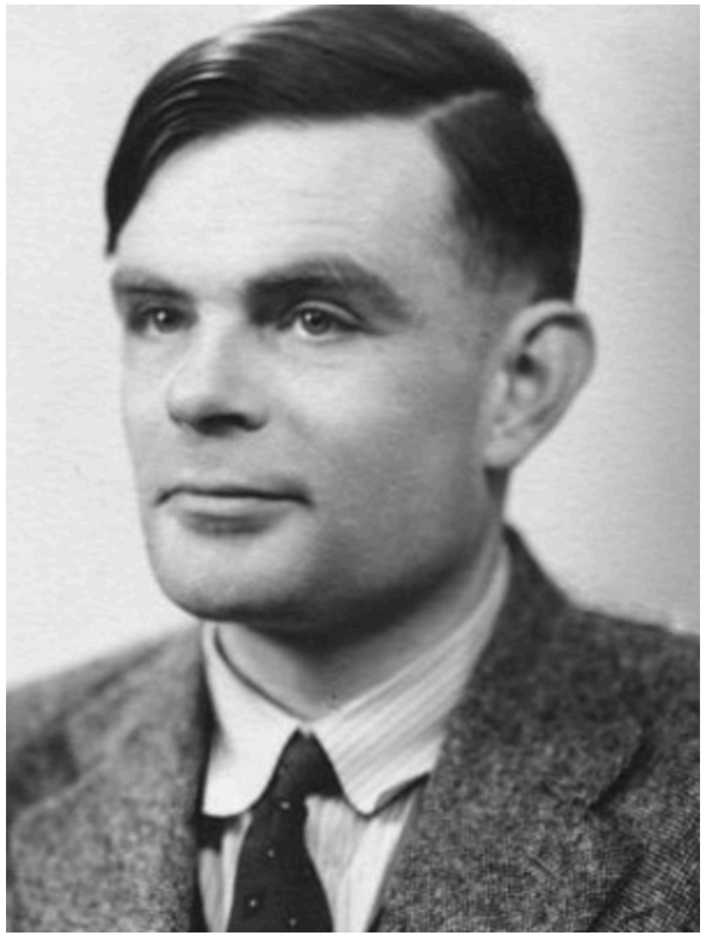
1. Developers are mostly comprehending/debugging code
2. Software ecosystems are growing in complexity
3. CPU hardware performance is stalling
4. Bugs aren’t getting any better

1. How did we get here?

2. What is the way out?



Alan Turing



"On Computable Numbers" 1937

Alonzo Church

"A set of postulates for the foundation of logic", 1932

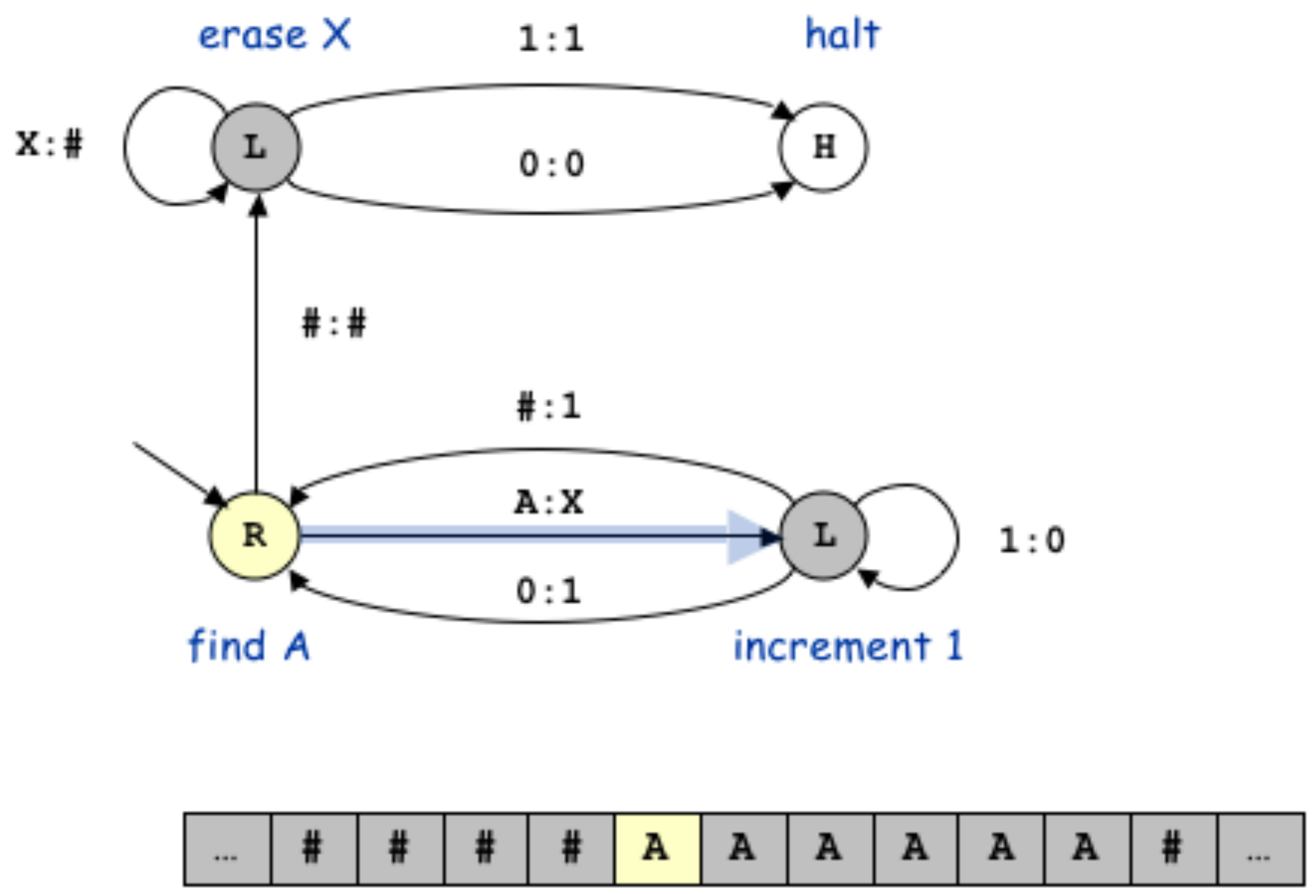
Lambda calculus

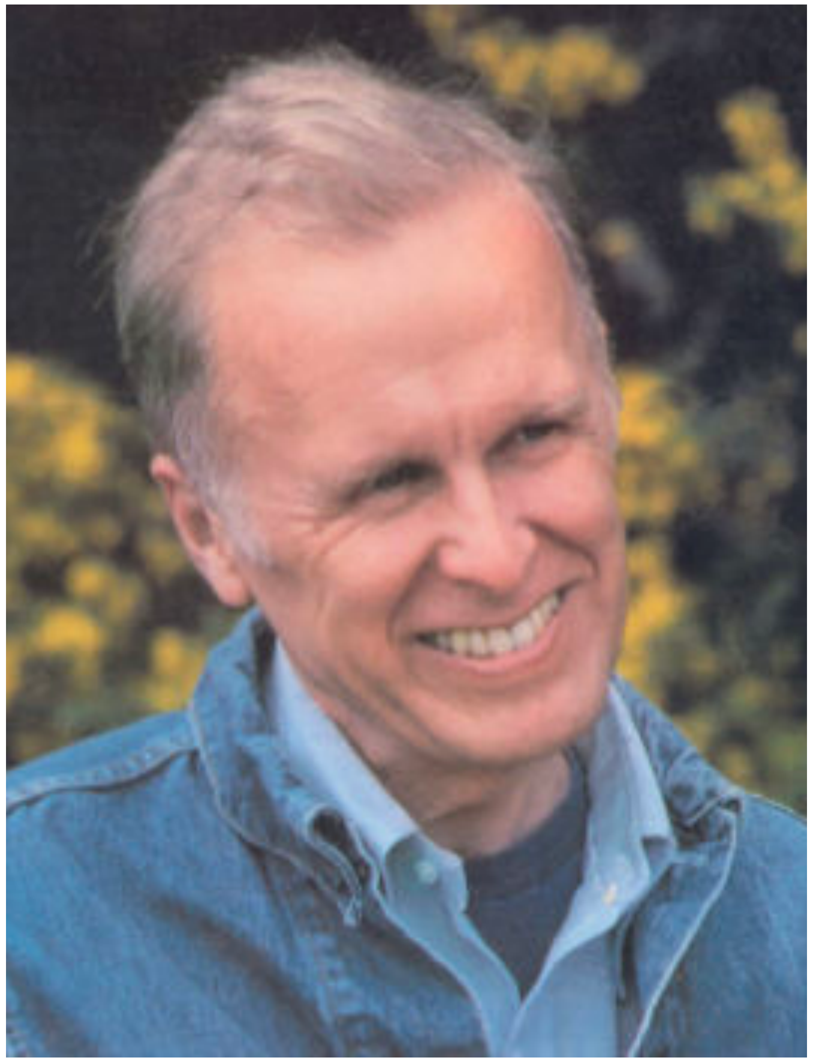
$$\begin{aligned}
[x \to y]\, x &= y \\
[x \to y]\, z &= z \\
[x \to y]\, \lambda\, z\, .\, x &= \lambda\, z\, .\, y \\
[x \to y]\, \lambda\, y\, .\, x\, y &= \lambda\, y'\, .\, y\, y' \\
[x \to y]\, x\, (\lambda\, x\, .\, x) &= y\, (\lambda\, x\, .\, x)
\end{aligned}$$

Lambda calculus

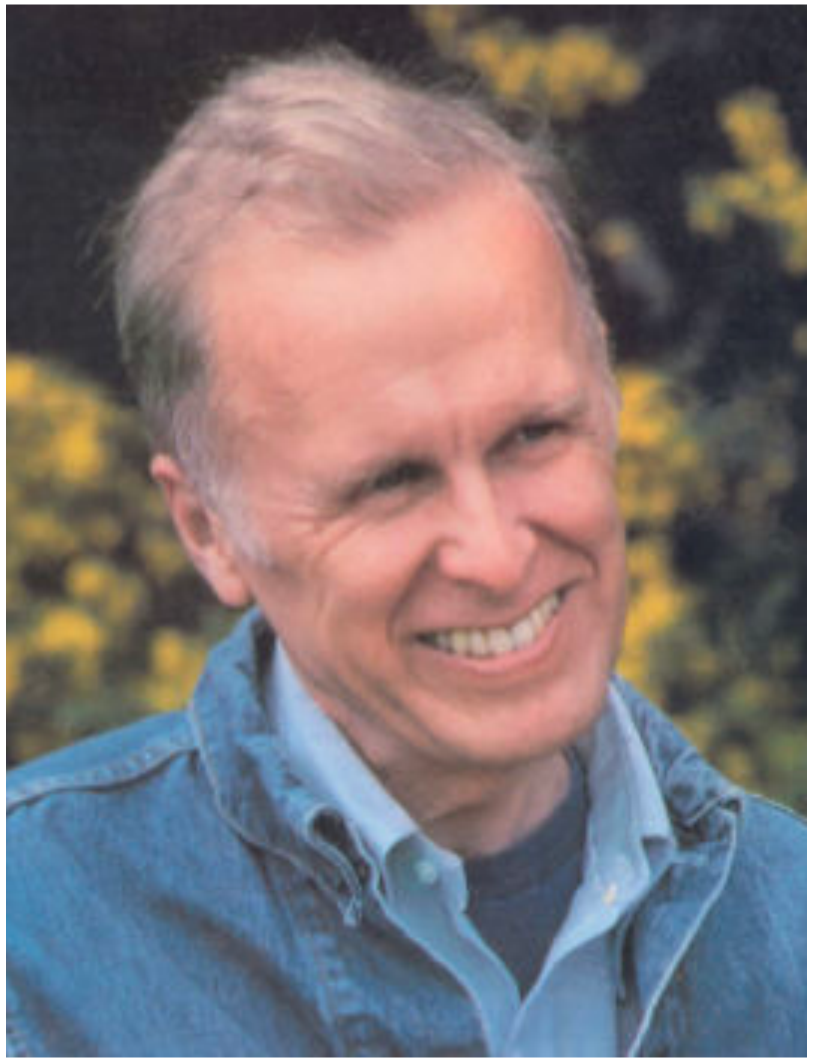
$$\begin{aligned}
[x \to y]\; x &= y \\
[x \to y]\; z &= z \\
[x \to y]\; \lambda\, z\, .\; x &= \lambda\, z\, .\; y \\
[x \to y]\; \lambda\, y\, .\; x\, y &= \lambda\, y'\, .\; y\, y' \\
[x \to y]\; x\, (\lambda\, x\, .\; x) &= y\, (\lambda\, x\, .\; x)
\end{aligned}$$

Turing machines





John Backus,
"History of FORTRAN" 1978



John Backus,
"History of FORTRAN" 1978

Prior to FORTRAN, most source language operations were not machine operations. Large inefficiencies in looping and computing addresses were masked by time spent in floating point subroutines.

The advent of the 704 with built-in floating point and indexing radically altered the situation. …It increased the problem of generating efficient programs by an order of magnitude by speeding up floating point operations by a factor of ten and thereby leaving inefficiencies nowhere to hide. This caused us to regard the design of the translator as the real challenge, not the simple task of designing the language.

Alan Perlis,
"The American Side of the Development of Algol" 1978

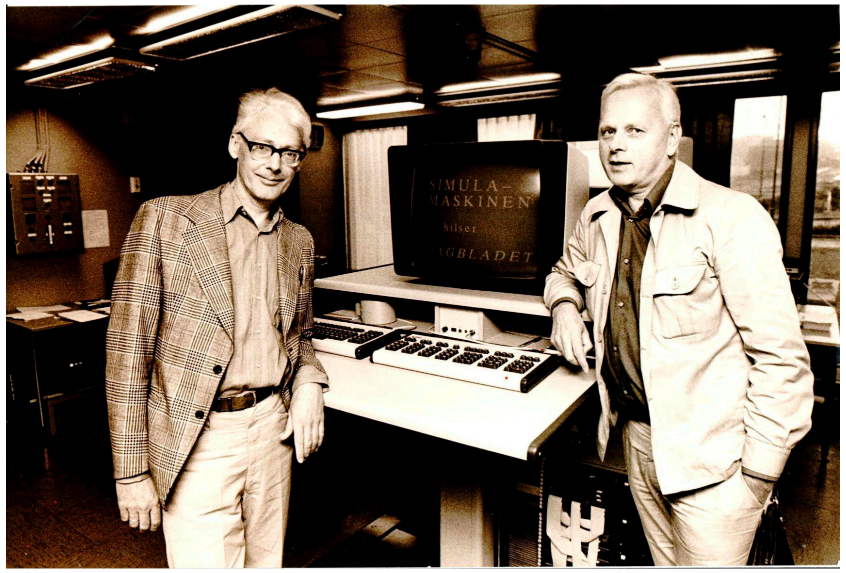
Algol introduced into programming languages such terms as type, declaration, identifier, for statement, while, if then else, switch, the begin end delimiters, block, call by value and call by name, typed procedures, declaration scope, dynamic arrays, side effects, global and local variables.

Algol was strongly derived from FORTRAN and its contemporaries. The logic, arithmetic and data organizations were close to those then being designed into real computers. Certain simple generalizations of computer instructions such as switch, for statement, and if statements were included because their semantics and computer processing were straight-forward consequences of single statement processing.

Ken Thompson,
Dennis Ritchie
“The Development of the C Language” 1993

C fits firmly in the traditional procedural family typified by Fortran and Algol 60. It is ‘close to the machine’ in that the abstractions it introduces are readily grounded in the concrete data types and operations supplied by conventional computers.

The most important [historical accident] has been the tolerance of C compilers to errors in type. C evolved from typeless languages. It did not suddenly appear to its earliest users as an entirely new language with its own rules; instead we continually had to adapt existing programs as the language developed, and make allowance for an existing body of code.



Kristen Nygaard and Ole-Johan Dahl
“The Development of SIMULA Languages”, 1978

Our first main task was to carry out resonance absorption calculations related to the construction of Norway's first nuclear reactor. Monte Carlo simulation methods were successfully introduced instead in 1949-1950. The necessity of using simulation and the need of a language for system description was the direct stimulus for SIMULA.

When writing simulation programs we had observed that processes often shared a number of common properties, both in data attributes and actions, but were structurally different in other respects so that they had to be described by separate declarations. Such partial similarity fairly often applied to processes in different simulation models, indicating that programming effort could be saved by somehow preprogramming the common properties.

Bjarne Stroustrup
"A History of C++: 1979-1991", 1993

[Simula's] class concept allowed me to map my application concepts into the language constructs in a direct way. The way Simula classes can act as coroutines made the inherent concurrency of my application easy to express.

The implementation of Simula, however, did not scale in the same way and as a result the whole project came close to disaster. The cost arose from several language features and their interactions: run-time type checking, guaranteed initialization of variables, concurrency support, and garbage collection of both user-allocated objects and procedure activation records.

Sun Microsystems, 1997

The team originally considered using C++, but rejected it for several reasons. They decided that C++'s complexity led to developer errors. The language's lack of garbage collection meant that programmers had to manually manage system memory, a challenging and error-prone task. The team also worried about the C++ language's lack of portable facilities for security, distributed programming, and threading. Finally, they wanted a platform that would port easily to all types of devices.

– Wikipedia??

Guido van Rossum
“The Making of Python”, 2003

My initial goal for Python was to serve as a second language for people who were C or C++ programmers, but who had work where writing a C program was just not effective.

Maybe it was something you'd do only once. It was the sort of thing you'd prefer to write a shell script for, but when you got into the writing details, you found that the shell was not the ideal language–you needed more data structures, more namespaces, or maybe more performance. The first sound bite I had for Python was, "Bridge the gap between the shell and C."

Turing languages

1957 – FORTRAN

1959 – ALGOL

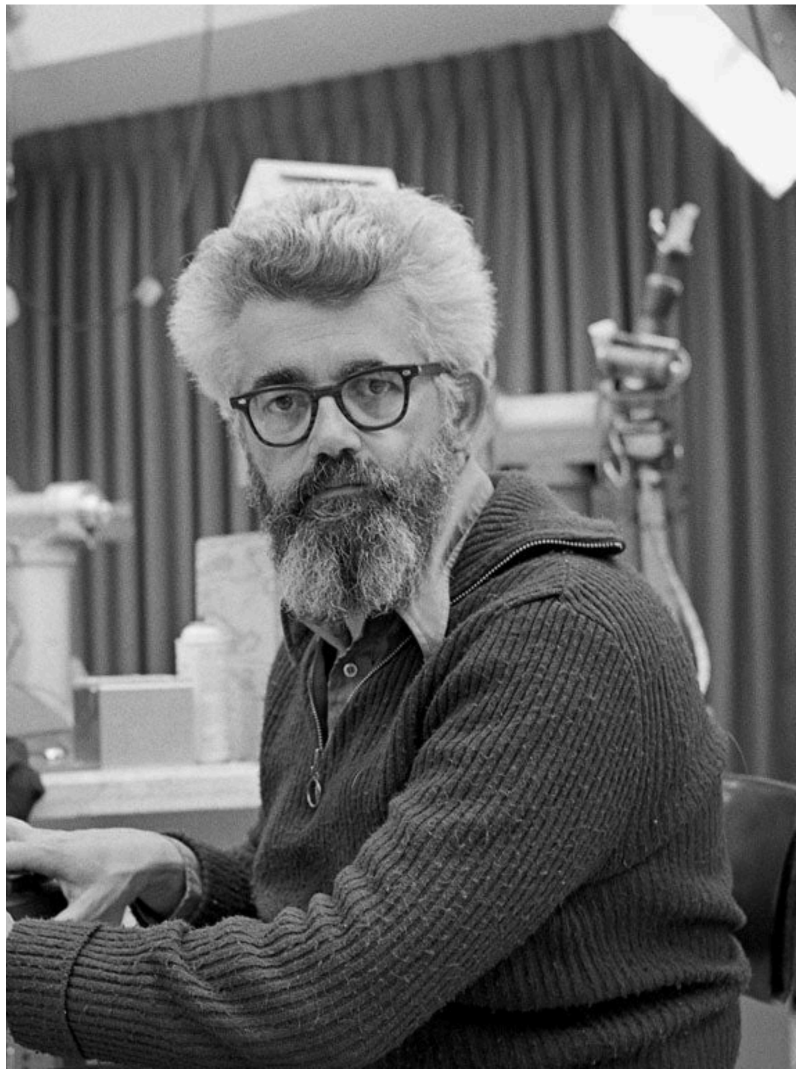
1962 – SIMULA

1972 – C

1979 – C++

1991 – Python

1995 – Java


John McCarthy,
"History of LISP" 1978

My own research in artificial intelligence [in 1958]… involved representing information about the world by sentences in a suitable formal language and a reasoning program that would decide what to do by making logical inferences. Representing sentences by list structure seemed appropriate and a list-processing language also seemed appropriate for programming the operations involved in deduction.

…One needs a notation for functions, and it seemed natural to use the λ-notation of Church (1941). I didn't understand the rest of his book, so I wasn't tempted to try to implement his more general mechanism for defining functions.

Peter Landin,
“The Next 700 Programming Languages” 1966

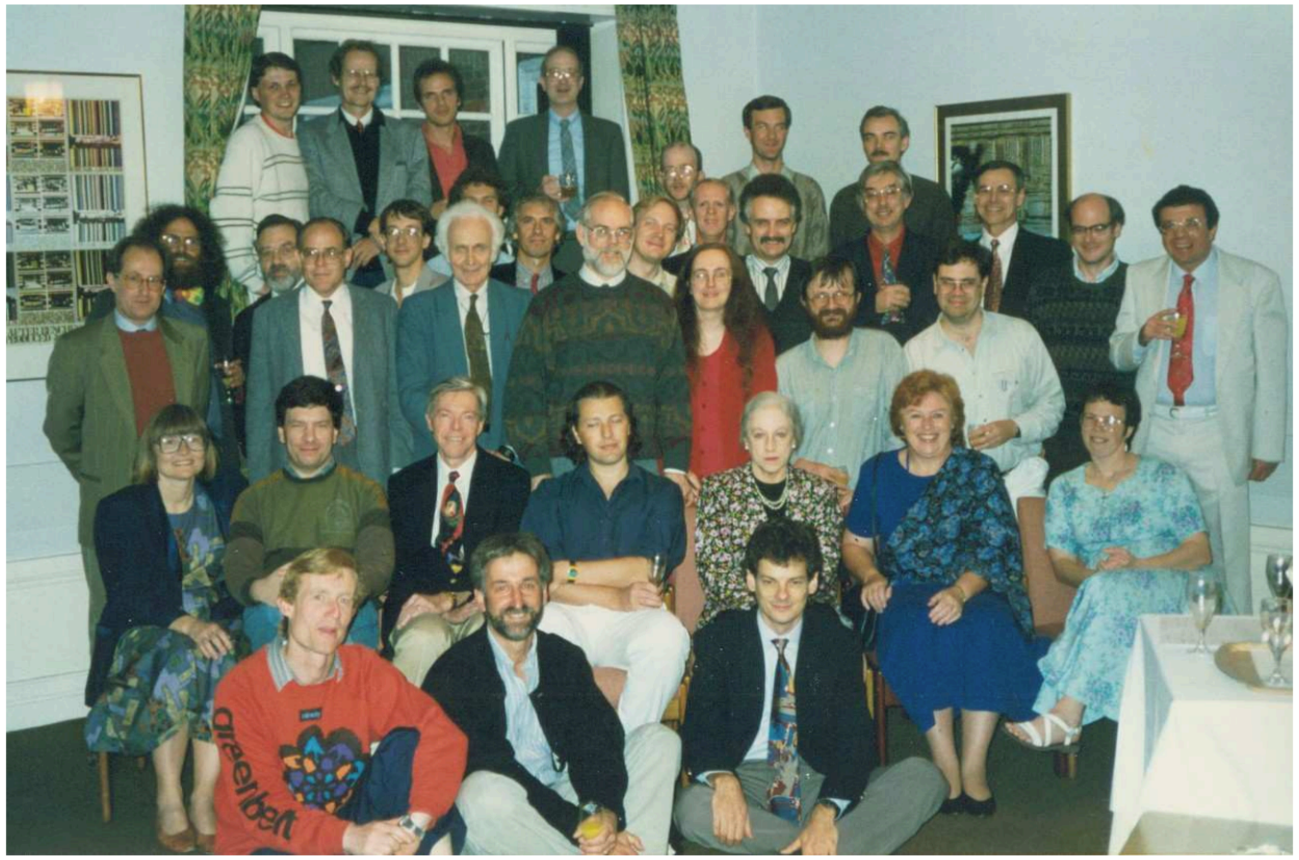
The languages people use to communicate with computers differ in their intended aptitudes, towards either a particular application area, or a particular phase of computer use (high level programming, program assembly, job scheduling, etc). The question arises, do the idiosyncrasies reflect basic logical properties of the situations that are being catered for? Or are they accidents of history and personal background that may be obscuring fruitful developments?

ISWIM is an attempt at a general purpose system for describing things in terms of other things, that can be problem-oriented by appropriate choice of "primitives." A possible first step in the research program is 1700 doctoral theses called "A Correspondence between *x* and Church's λ-notation.”

Robin Milner,
"A Metalanguage for Interactive Proof in LCF"
1978

The principal aims in designing ML were to make it impossible to prove non-theorems yet easy to program strategies for performing proofs.

A strategy–or recipe for proof–could be something like "induction on *f* and *g* , followed by assuming antecedents and doing case analysis, all interleaved with simplification". This is imprecise–analysis of what cases? what kind of induction, etc, etc.–but these in turn may well be given by further recipes, still in the same style.



Hudak et al.
"A History of Haskell: Being Lazy with Class"
2007

The simplicity and elegance of functional programming captivated the present authors. Lazy evaluation– with its direct connection to the pure, call-by-name lambda calculus, the remarkable possibility of representing and manipulating infinite data structures, and addictively simple and beautiful implementation techniques–was like a drug.

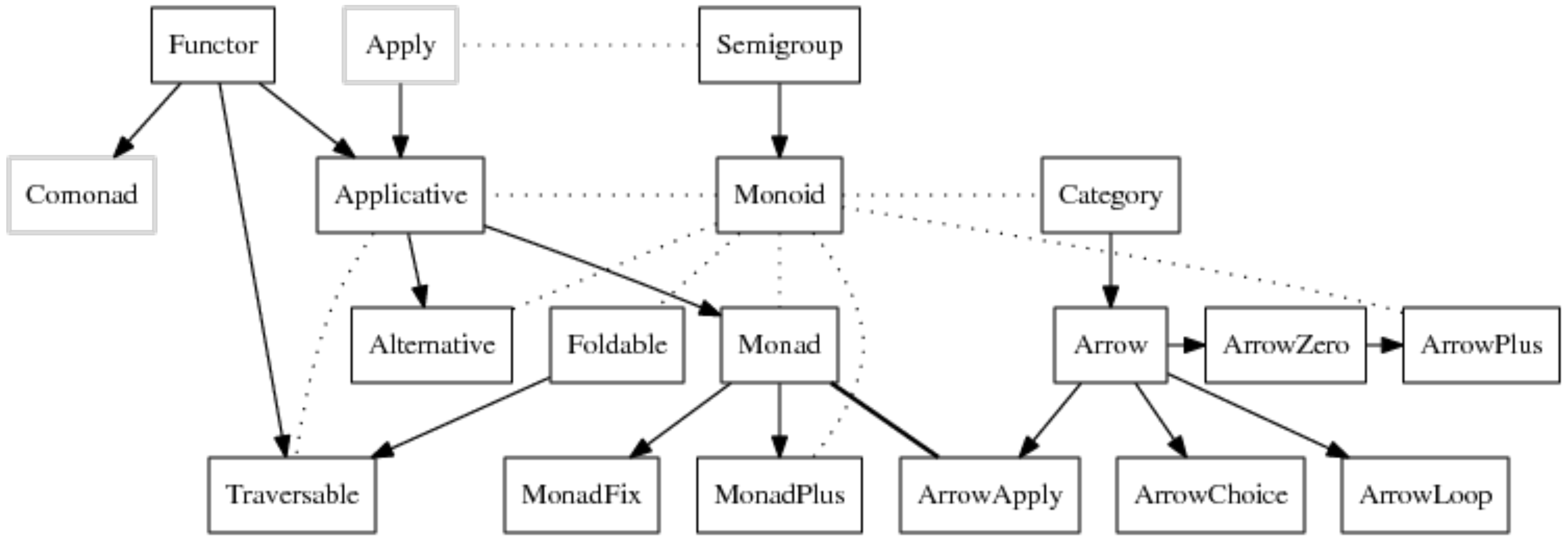
Turing languages

1957 – FORTRAN
1959 – COBOL, ALGOL
1962 – SIMULA
1972 – C, Smalltalk
1979 – C++
1991 – Python
1995 – Java

Church languages

1959 – LISP
1966 – ISWIM
1972 – Prolog
1978 – ML
1990 – Haskell

Church languages are intimidating



```
Couldn't match type `k0' with `b'
  because type variable `b' would escape its scope
This (rigid, skolem) type variable is bound by
  the type signature for
    groupBy :: Ord b => (a -> b) -> Set a -> Set (b, [a])
The following variables have types that mention k0
```

PL theory can be really dense

$$
\frac{
\begin{array}{c}
\Sigma_0;\Gamma_1 \vdash A = \mathsf{D}\ \bar{v} : \mathsf{Set}_n \qquad \Gamma = \Gamma_1(x : A)\Gamma_2 \\
\mathsf{data}\ \mathsf{D}\ \Delta : \mathsf{Set}_n\ \mathsf{where}\ \mathsf{c_i}\ \Delta_i\ [\bar{j}_i \mid b_i] \in \Sigma_0 \qquad \exists k.\ [\bar{j}_k \mid b_k] \neq [] \\[1ex]
\left(
\begin{array}{c}
\Delta'_i = \Delta_i(\bar{j}_i : \mathbb{I})[\bar{v} \,/\, \Delta] \qquad \bar{q}_i = \hat{\Delta}_i[\bar{j}_i \mid b_i][\bar{v} \,/\, \Delta] \\
\rho_i = \mathbb{1}_{\Gamma_1} \uplus [\mathsf{c_i}\ \bar{q}_i \,/\, x] \qquad \rho'_i = \rho_i \uplus \mathbb{1}_{\Gamma_2} \\
\Theta_i = \textsc{boundary}(\bar{j}_i);\Theta \\
\Sigma_{i-1};\Gamma_1\Delta'_i(\Gamma_2\rho_i) \vdash \mathsf{f}\ \bar{q}\rho'_i := Q_i : C\rho'_i \mid \Theta_i \rightsquigarrow \Sigma_i
\end{array}
\right)_{i=1\ldots n} \\[1ex]
\Delta_{\mathsf{hc}} = (r : \mathbb{I})(u : \mathbb{I} \to \mathsf{Partial}\ r\ (\mathsf{D}\ \bar{v}))(u_0 : \mathsf{D}\ \bar{v}\ [\, r \mapsto u\ \mathsf{i0}\,]) \\
\rho_{\mathsf{hc}} = \mathbb{1}_{\Gamma_1} \uplus [\mathsf{hcomp}\ r\ u\ u_0 \,/\, x] \qquad \rho'_{\mathsf{hc}} = \rho_{\mathsf{hc}} \uplus \mathbb{1}_{\Gamma_2} \\
\Sigma_n;\Gamma_1\Delta_{\mathsf{hc}}(\Gamma_2\rho_{\mathsf{hc}}) \vdash \mathsf{f}\ \bar{q}\rho'_{\mathsf{hc}} := Q_{\mathsf{hc}} : C\rho'_{\mathsf{hc}} \mid (r = 1);\Theta \rightsquigarrow \Sigma_{n+1}
\end{array}
}{
\Sigma_0;\Gamma \vdash \mathsf{f}\ \bar{q} := \mathsf{case}_x \left\{
\begin{array}{l}
\mathsf{c_1}\ \bar{q}_1 \mapsto Q_1; \ldots; \mathsf{c_n}\ \bar{q}_n \mapsto Q_n \\
\mathsf{hcomp}\ r\ u\ u_0 \mapsto Q_{\mathsf{hc}}
\end{array}
\right\} : C \mid \Theta \rightsquigarrow \Sigma_{n+1}
}
$$

Vezzosi et al. "Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types" ICFP '19

Thesis:

Ideas from Church languages will radically improve how we 1) think, 2) communicate, and 3) program in software design.

Corollary:

Learning these ideas gives you a *programming superpower* to make correct and reliable software way more productively.

Von Neumann languages (e.g., the FORTRANs, the ALGOLs) constantly keep our noses pressed in the dirt of the separate computation of single words, whereas we should be focusing on the form and content of the overall result. <u>We regard the DO, FOR, WHILE statements and the like as powerful tools, whereas they are in fact weak palliatives</u> that are necessary to make the primitive von Neumann style of programming viable at all.

… While it was perhaps natural and inevitable that languages like FORTRAN and its successors developed out of the von Neumann computer, that such languages have dominated our thinking for twenty years is unfortunate. Their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer <u>far greater intellectual and computational power.</u>

– John Backus, "The history of FORTRAN I, II, and III", 1978

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] ∗ b[i]
  return c
```

- Statements operate on invisible state
- Computes word-at-a-time by repetition of assignment/modification
- Requires names for arguments, iterator, return value

```
let innerproduct = zip |> (map ∗) |> (reduce +)
```

- Built from composable functions (map, reduce, pipe)
- Operates on whole conceptual units (lists), no repeated steps
- No names for arguments or temporaries

Tackling complexity through modularity

Modular design is the key to successful programming. When writing a modular program to solve a problem, one first divides the problem into subproblems, then solves the subproblems, and finally combines the solutions.

The ways in which one can divide up the original problem depend directly on the ways in which one can glue solutions together. Therefore, to increase one's ability to modularize a problem conceptually, one must provide new kinds of glue in the programming language.

– John Hughes, "Why Functional Programming Matters" 1989

```
fun innerproduct(a, b, n):
  c := 0
  for i := 1 step 1 until n do
    c := c + a[i] * b[i]
  return c
```

```
let innerproduct = zip |> (map *) |> (reduce +)
```

Python

```
# Any * Any -> Any
def innerproduct(a, b): ...
```

C

```
typedef struct { float* arr; ... } vec_t;
float innerproduct(vec_t a, vec_t b)
```

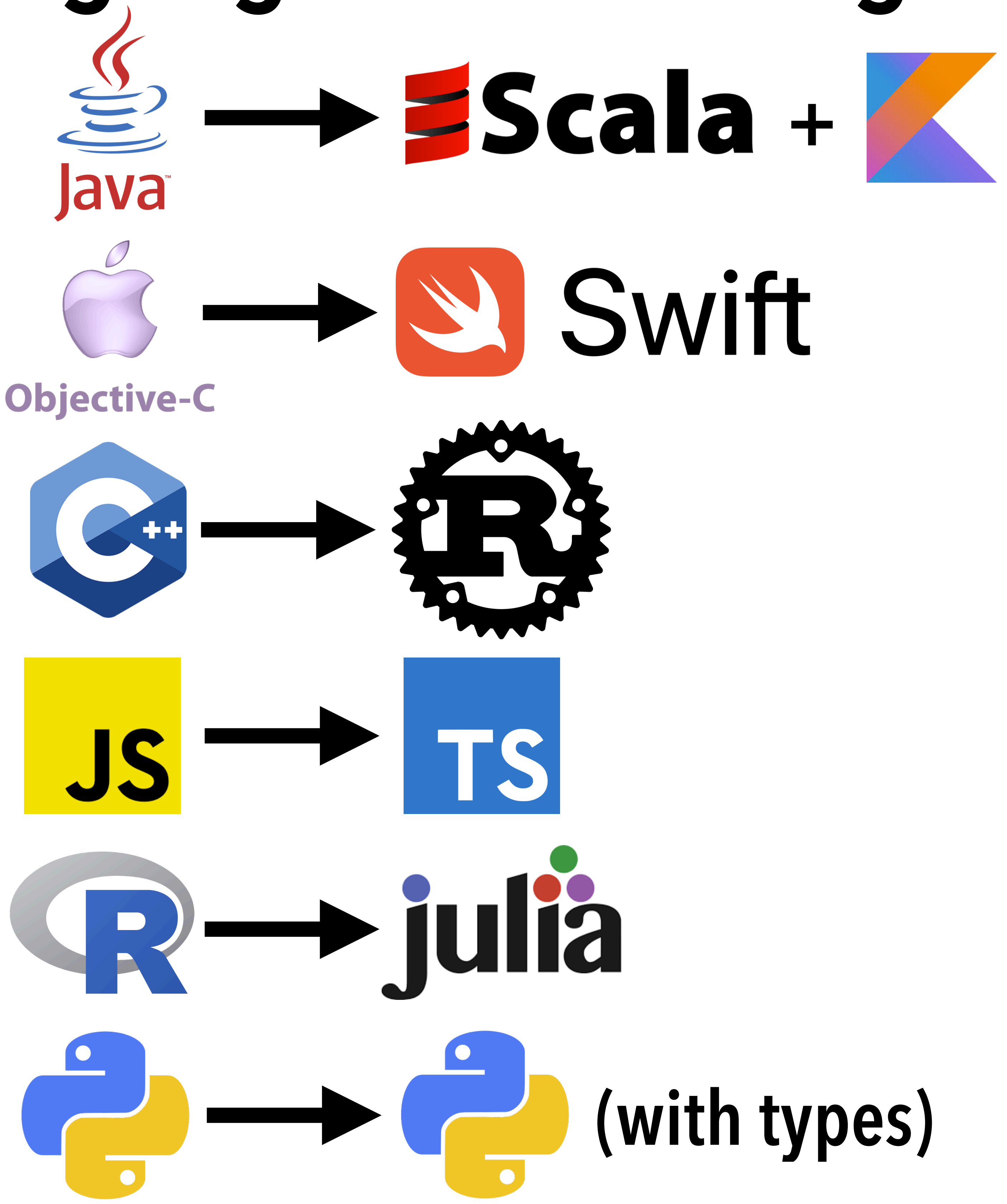
Java

```
public class Vec<T> {
  public T innerproduct(Vec<T> other);
}
```

Idris

```
innerproduct : Vec n a -> Vec n a -> a
```

Church languages are coming!



Companies are realizing FP benefits

Messenger used to receive bugs reports on a daily basis; since the introduction of Reason, there have been a total of 10 bugs (that's during the whole year, not per week)! Refactoring speed went from days to hours to dozens of minutes.

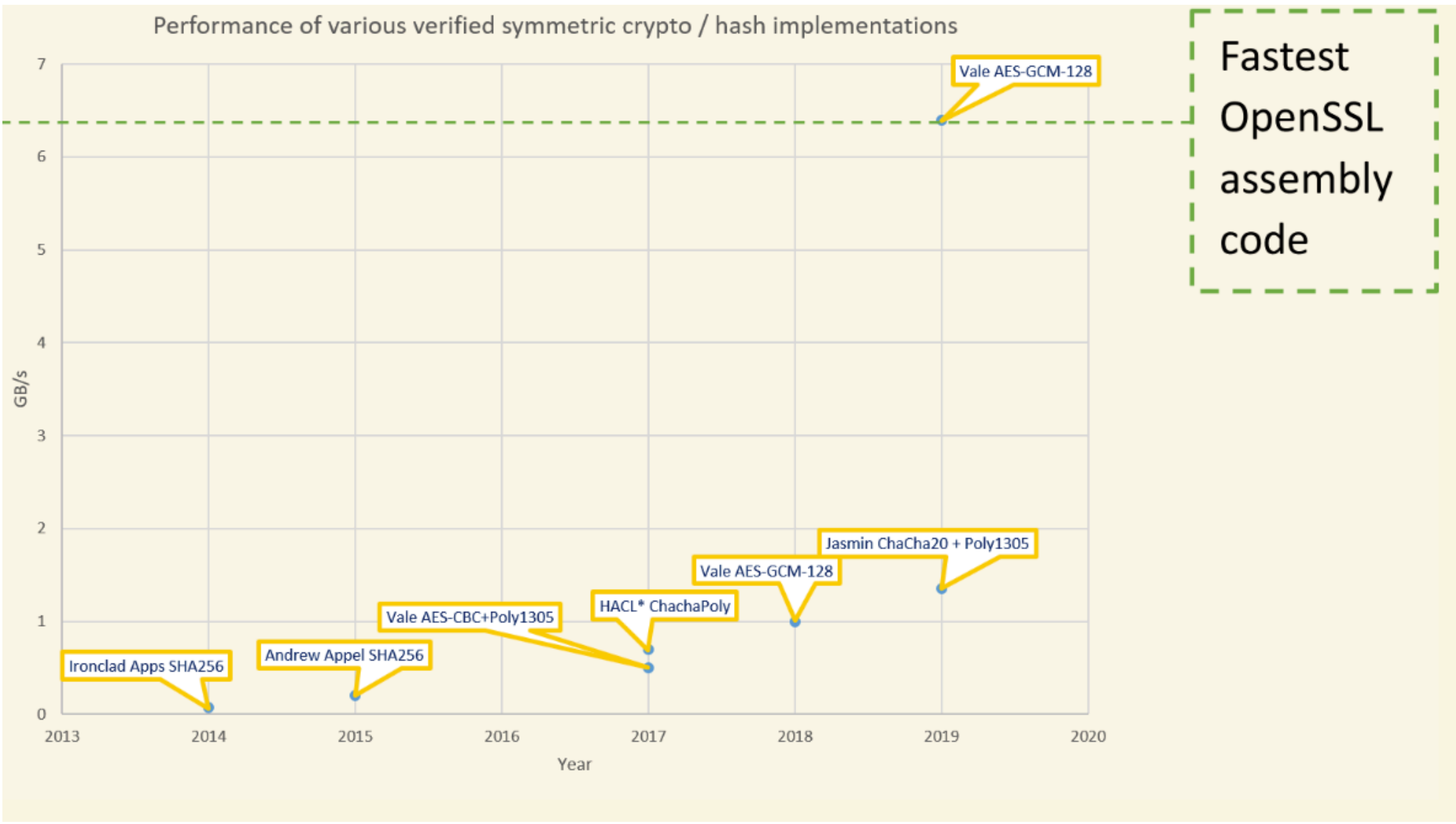
– "Messenger.com Now 50% Converted to Reason" 2017

Being able to encode constraints of your application in the type system makes it possible to refactor, modify, or replace large swaths of code with confidence. Rust's error model forces developers to handle every corner case. [Our system] needs very little attention. We were able to leave it running without any issues through the holiday break.

– "Rust at OneSignal" 2017

Formal verification is around the corner

```
val aes_ctr_encrypt_bytes:
    v:variant
  -> key:aes_key v
  -> n_len:size_nat{n_len <= 16}
  -> nonce:lbytes n_len
  -> c:size_nat
  -> msg:bytes{length msg / 16 + c <= max_size_t} ->
  Tot (ciphertext:bytes{length ciphertext == length msg})
```


Performance of various verified symmetric crypto / hash implementations
7
6
5
4
3
2
1
0
GB/s
2013
2014
2015
2016
2017
2018
2019
2020
Year
Vale AES-GCM-128
Jasmin ChaCha20 + Poly1305
Vale AES-GCM-128
HACL* ChachaPoly
Vale AES-CBC+Poly1305
Andrew Appel SHA256
Ironclad Apps SHA256
Fastest OpenSSL assembly code

Takeaway points

- The software world is ready for better languages
  - The languages we use today have barely changed in 60 years
  - Developer spending 5% time writing code
  - Huge, buggy, slow software ecosystems

- Church languages focus on mathematics of programming
  - Advanced type systems and functional features promote modularity, specification, and verification
  - Not mainstream due to inefficiencies, lack of application focus

- The future is functional
  - Better compilers, better understanding of which concepts matter in practice, better awareness are moving Church languages to mainstream

Key concepts and skills: theory

- Formal representation of and reasoning about programs
  - Descriptions of syntax and semantics of languages
  - Mathematical specification of program behavior (type systems)
  - Inductive proofs about programs and languages
  - Formal model of both functional and imperative languages
- You should be able to…
  - Learn more advanced PL theory
  - Read basic PL theory papers (and understand them)
  - Understand the essence of basic computational concepts

Key concepts and skills: practice

- Functional programming techniques
  - Explicit state-passing with expression-oriented programming
  - Abstract patterns with higher-order functions
  - Error handling and inductive data with algebraic data types
  - Generic/modular programming with polymorphism and traits
  - Resource management/concurrency with linear types
  - Verified communication with session types

- You should be able to…
  - Structure low-level code to avoid large classes of bugs
  - Learn new functional programming languages more effectively
  - Recognize opportunities for Church programming in Turing languages
    - "Will, I feel guilty now every time I write a `for` loop" – CURIS student in my office for the summer

Course prerequisites

- Theory: CS 103
  - First-order logic, induction, discrete math
  - Strongly encouraged
- Systems: CS 107 + 110
  - 107 absolutely
  - 110 for a bit of the multithreading

Course logistics

- **Weekly assignments**
  - Get started early!
  - Mixture of written problems and programming problems
  - Expect 15 hr/week including lecture

- **No midterm, take-home final exam**
  - Assignments are more work to compensate

- **Everything available on course website**
  - cs242.stanford.edu
  - Including detailed lecture notes and slides

Attending lectures

- Yes, everything is recorded through SCPD
  - Yes, you can watch everything from your sofa
- I still recommend you come to class!
  - Put yourself in a learning environment
  - I will always give time for thought and questions
- …But no laptops
  - Don’t distract people if you’re going to attend lectures