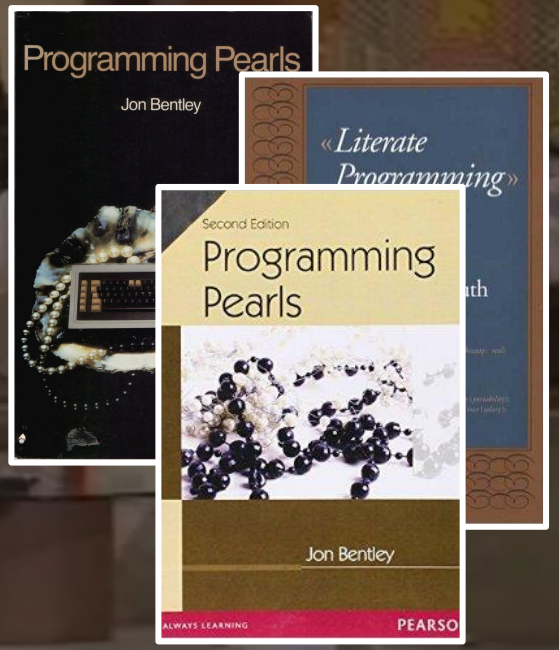
From Lone Dwarfs To Giant Superclusters: Rethinking OS Abstractions for the Cloud

Nikos Vasilakis, Ben Karel, Jonathan M. Smith
The University of Pennsylvania | HotOS 2015

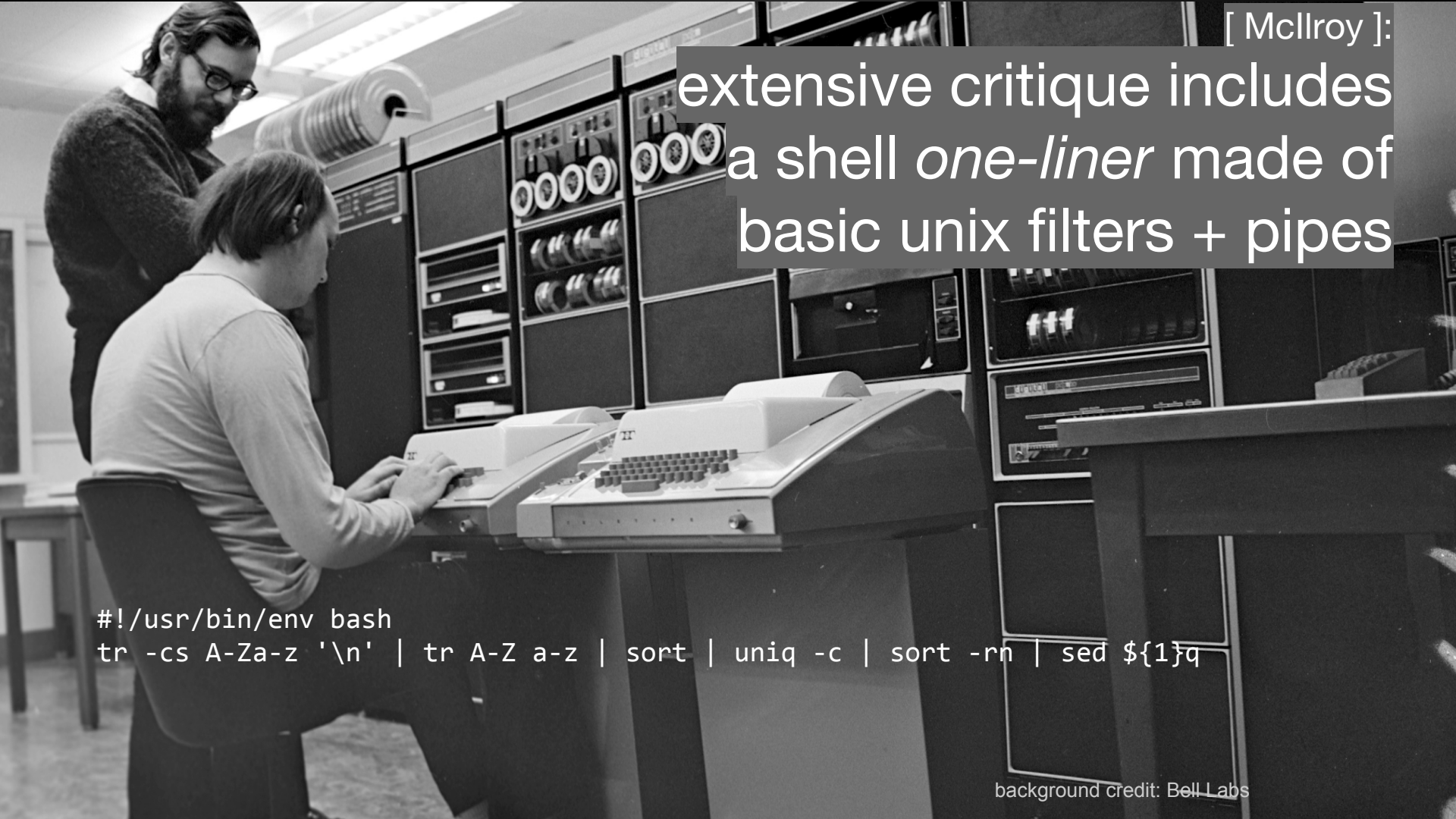
[Bentley]: programming pearls

"given a text file and an integer *k*, print the *k* most common words in the file (and the number of their occurrences) in decreasing frequency."

[Knuth]: a literate program



background credit: from the film "The Party", 1968


[McIlroy]:
extensive critique includes a shell *one-liner* made of basic unix filters + pipes
#!/usr/bin/env bash
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
background credit: Bell Labs

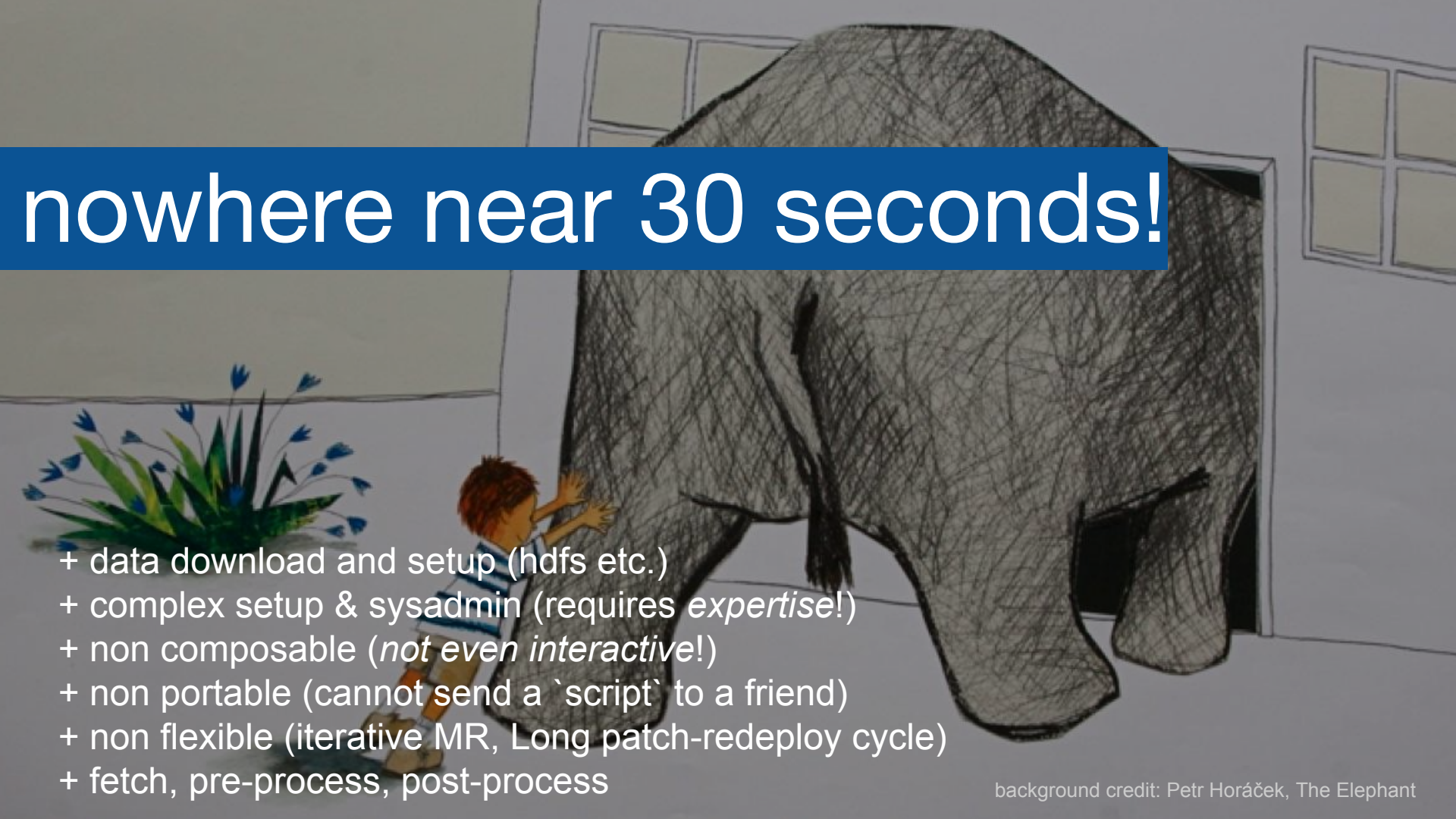
How would *we* solve it *today*?

background credit: Larry Salk, Summer Cocktail Party with English Butler, 1961

Hadoop

using
a 10-node
Amazon cluster

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```



nowhere near 30 seconds!

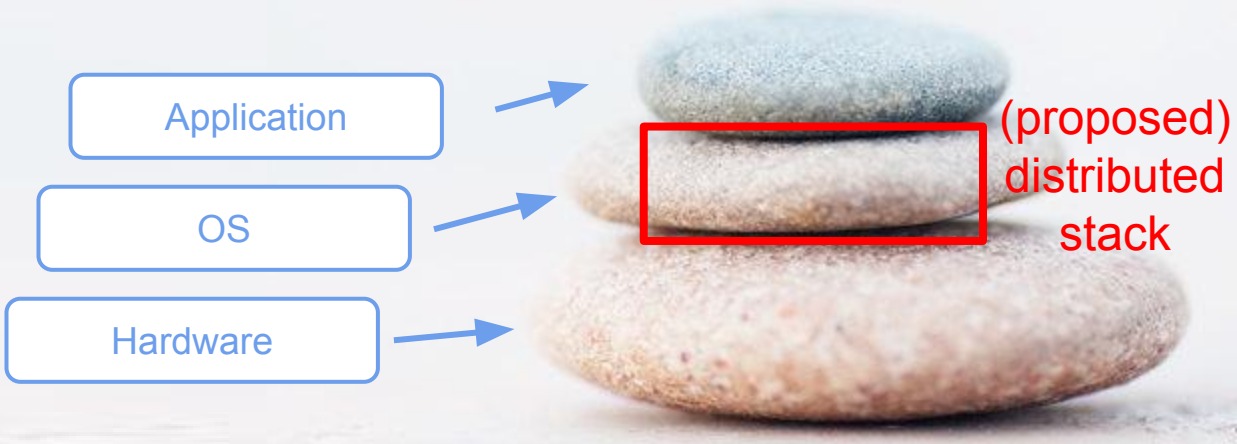
+ data download and setup (hdfs etc.)
+ complex setup & sysadmin (requires *expertise*!)
+ non composable (*not even interactive*!)
+ non portable (cannot send a `script` to a friend)
+ non flexible (iterative MR, Long patch-redeploy cycle)
+ fetch, pre-process, post-process

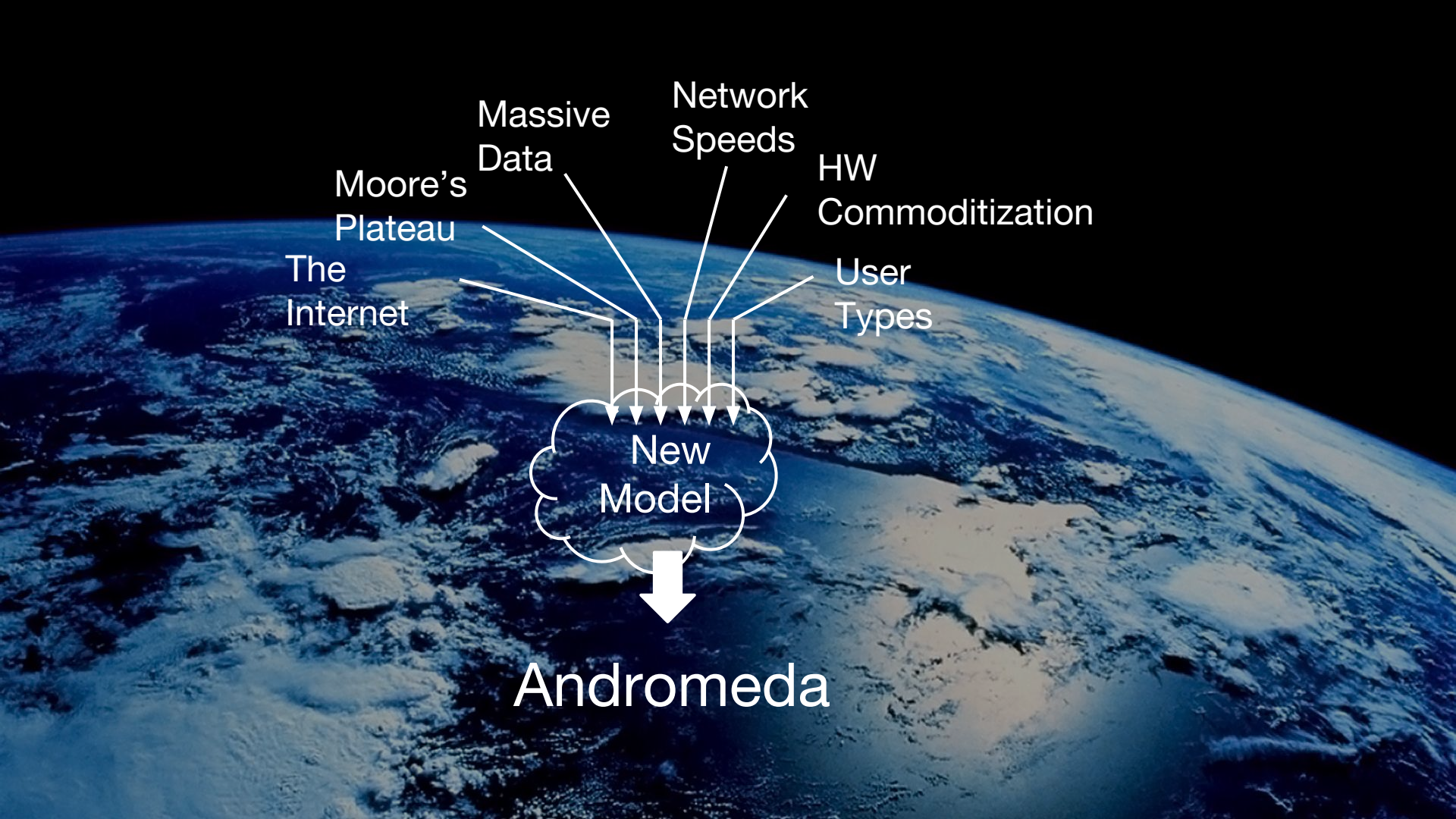
background credit: Petr Horáček, The Elephant

cloud software stack

transformation is long overdue

we have a set of ideas towards this direction




Massive Data
Network Speeds
Moore's Plateau
HW Commoditization
The Internet
User Types
New Model
Andromeda

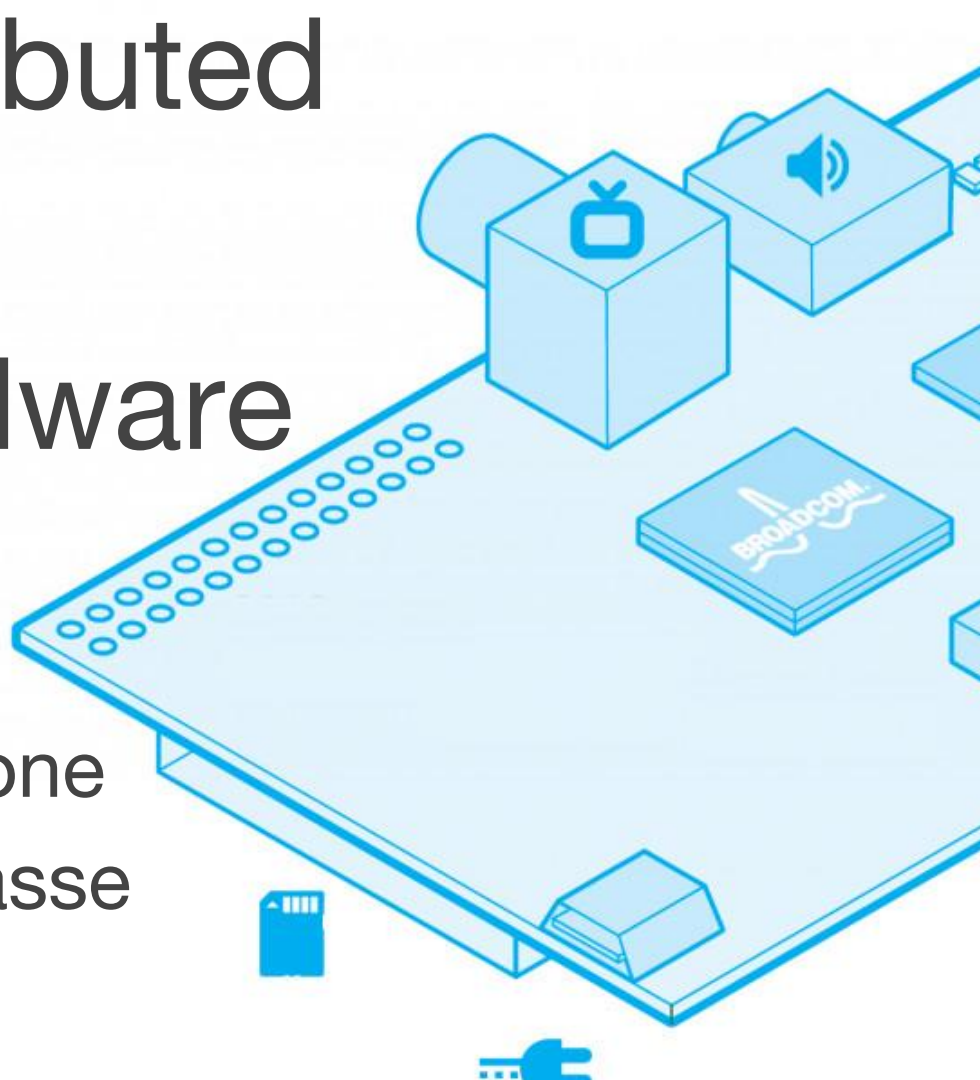

~~CS experts~~
business biology environmental studies communication military religion
chemistry education history engineering linguistics
mathematics space evolution physics anthropology archaeology
economics medicine sociology psychology earth sciences
One size does not fit all!

data deluge

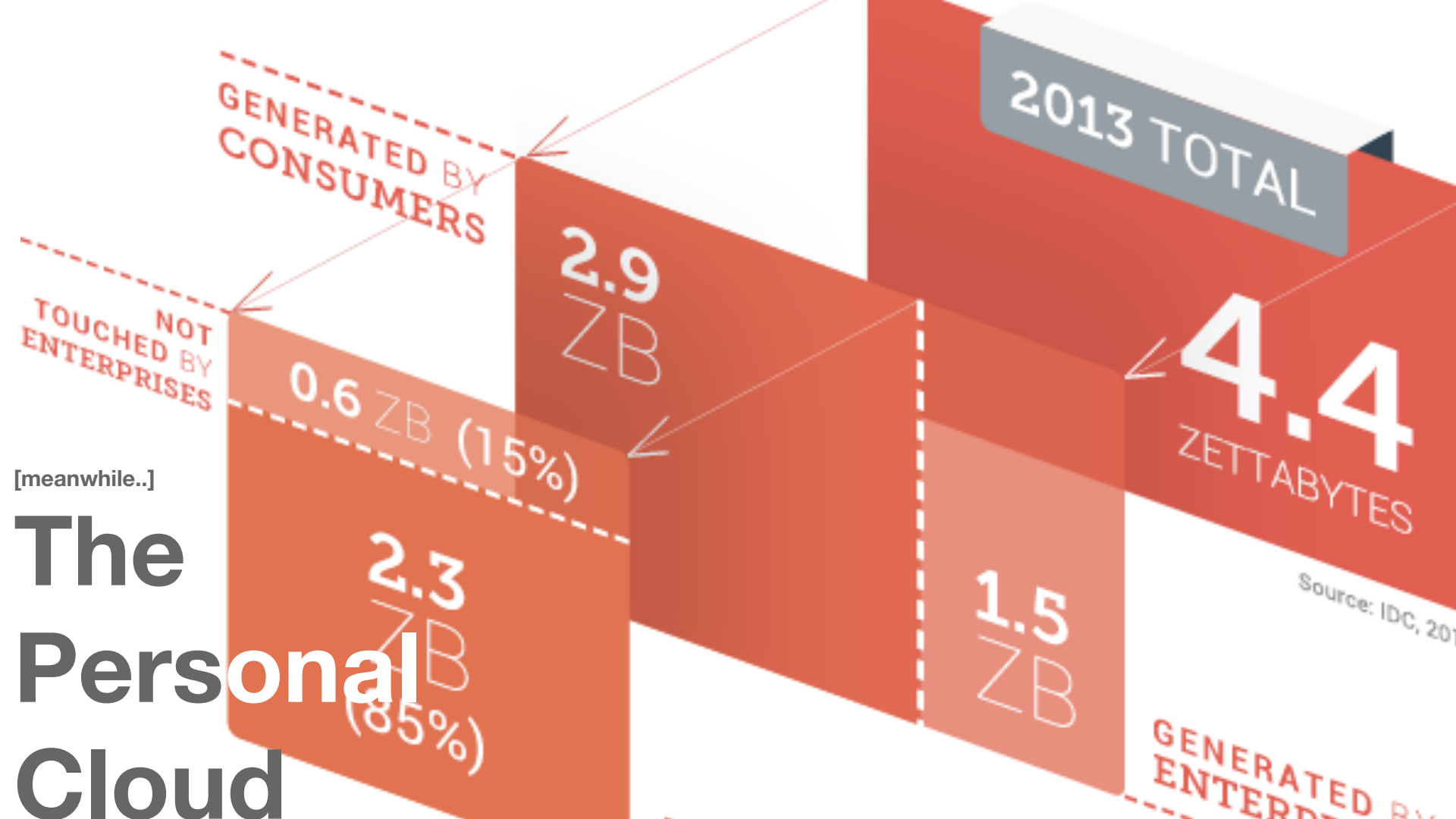
data processing at extraordinary scale
revolution of distributed processing frameworks

90% of data today
was created within the last 2 years!

ubiquity of distributed processing on commodity hardware

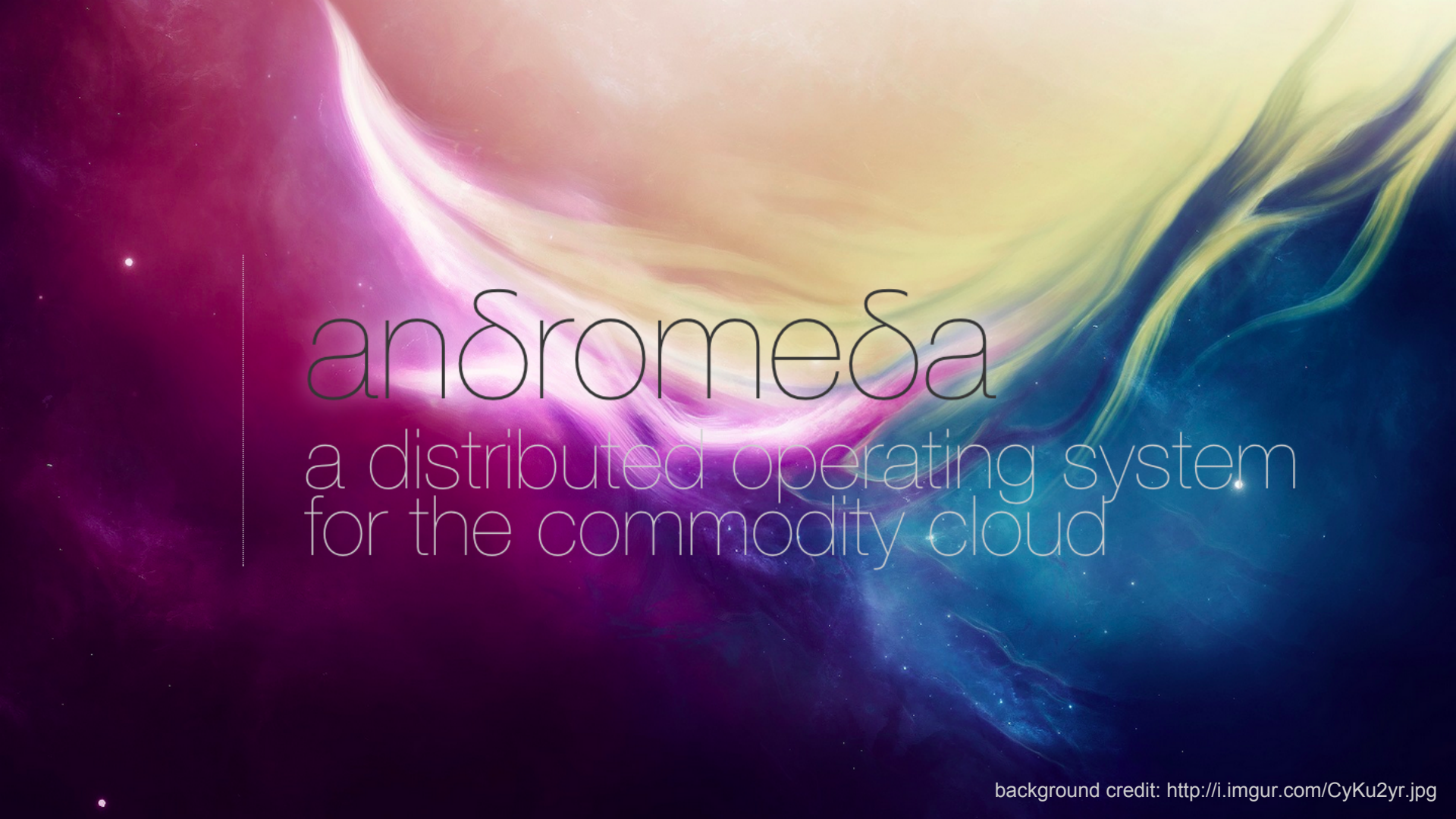


- heterogeneous, fault-prone
+ cheaper, tolerant en-masse

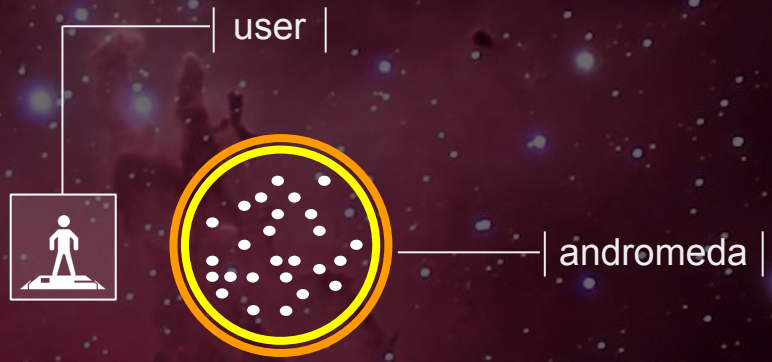


[meanwhile..]

The Personal Cloud


anδromeδa
a distributed operating system
for the commodity cloud
background credit: http://i.imgur.com/CyKu2yr.jpg

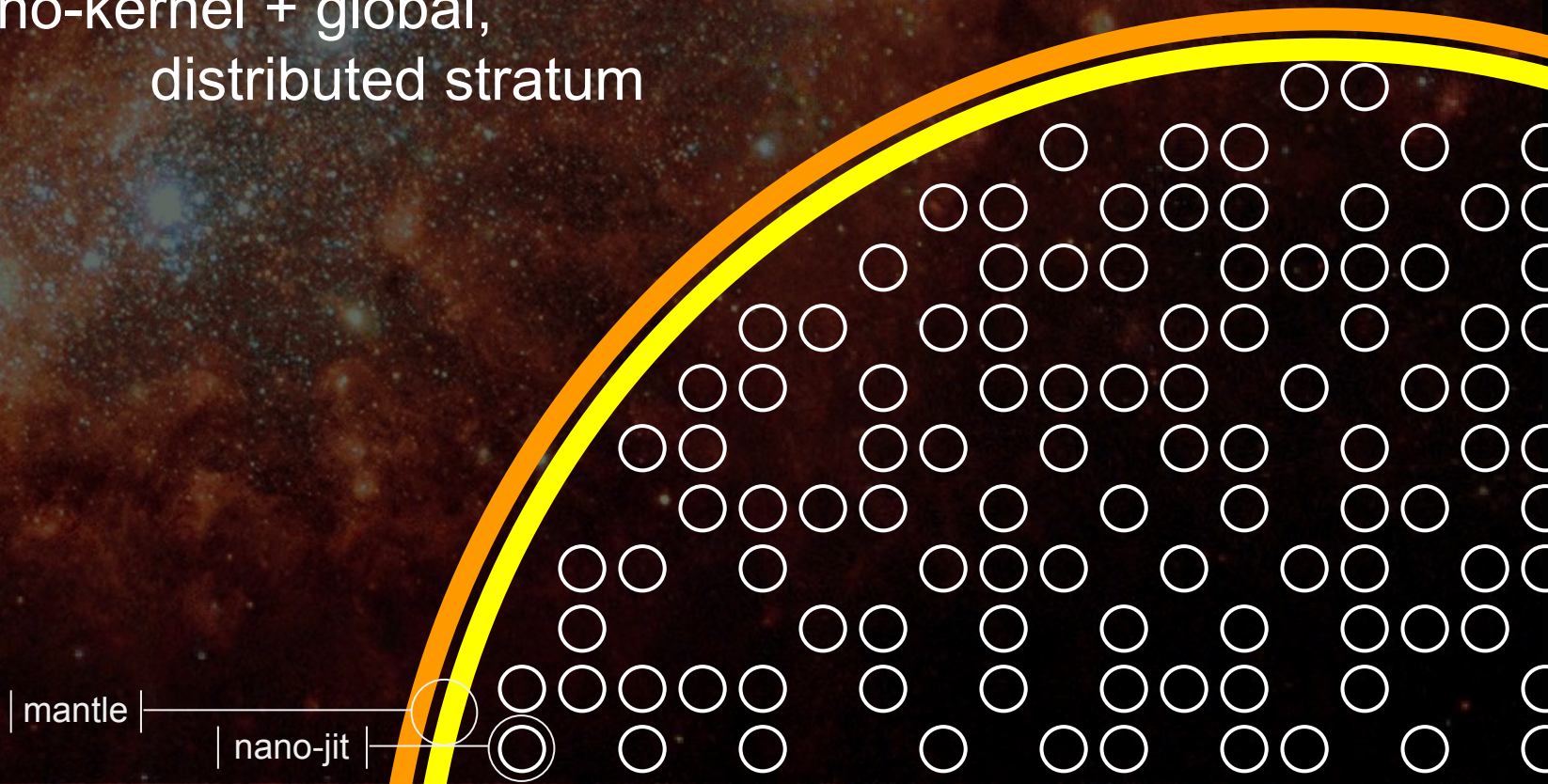
Andromeda Themes



Fully-transparent and homogenous distribution
Coupling of {linguistic features, systemic mechanisms}
Focus on fully-interactive use

Fully-Transparent Object Distribution

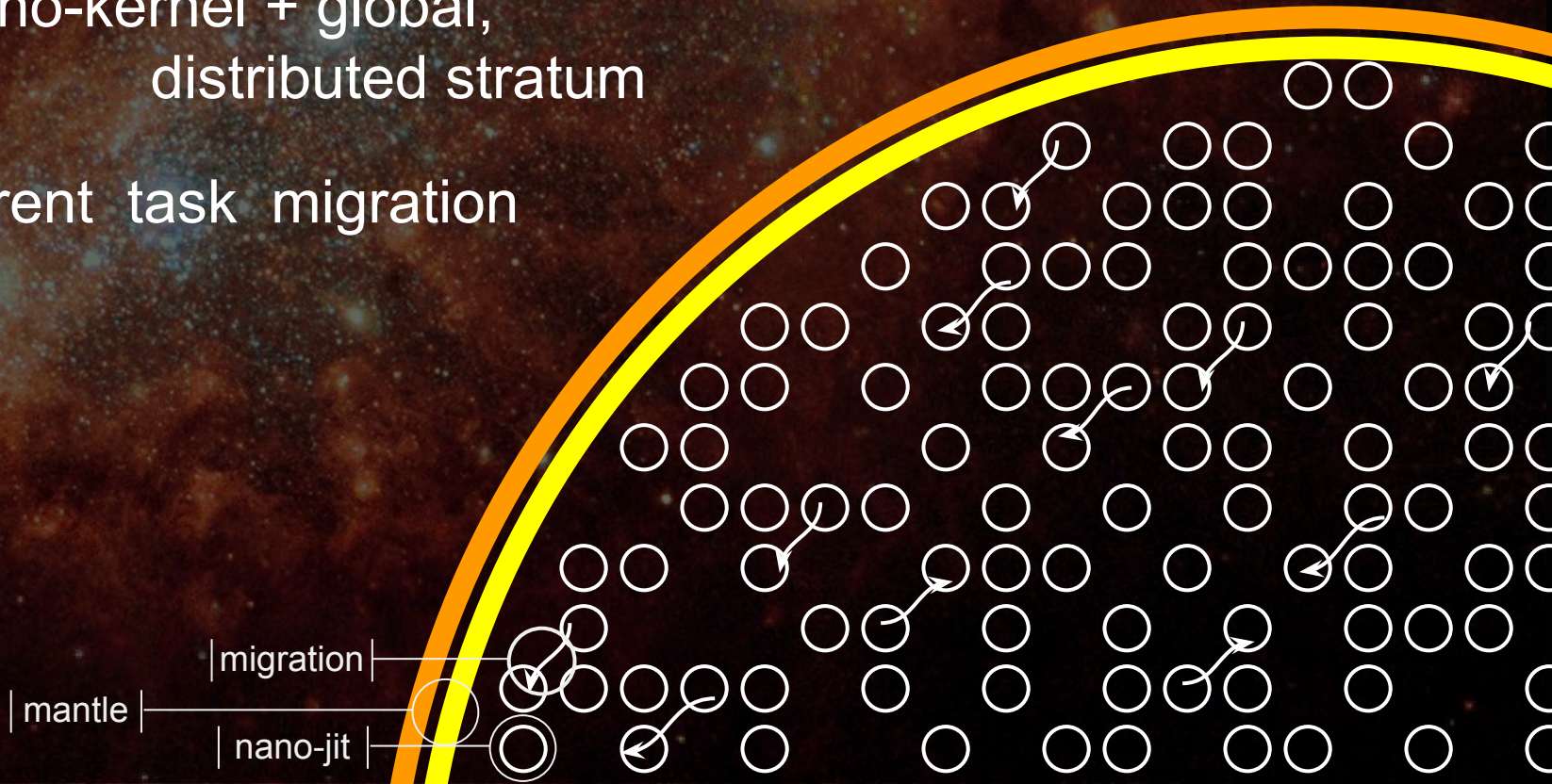
local nano-kernel + global,
distributed stratum



Fully-Transparent Distribution

local nano-kernel + global,
distributed stratum

transparent task migration

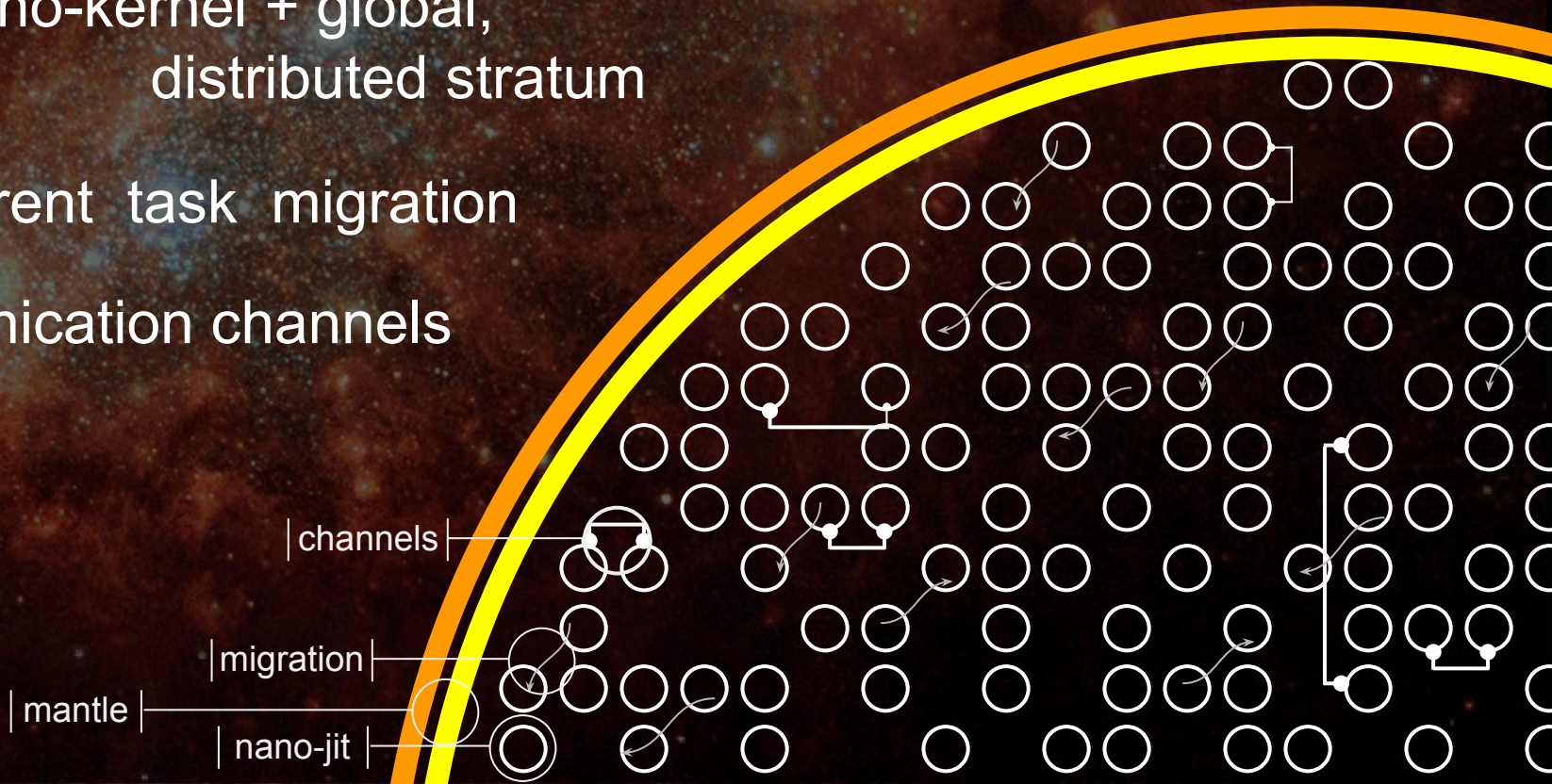


Fully-Transparent Distribution

local nano-kernel + global,
distributed stratum

transparent task migration

communication channels



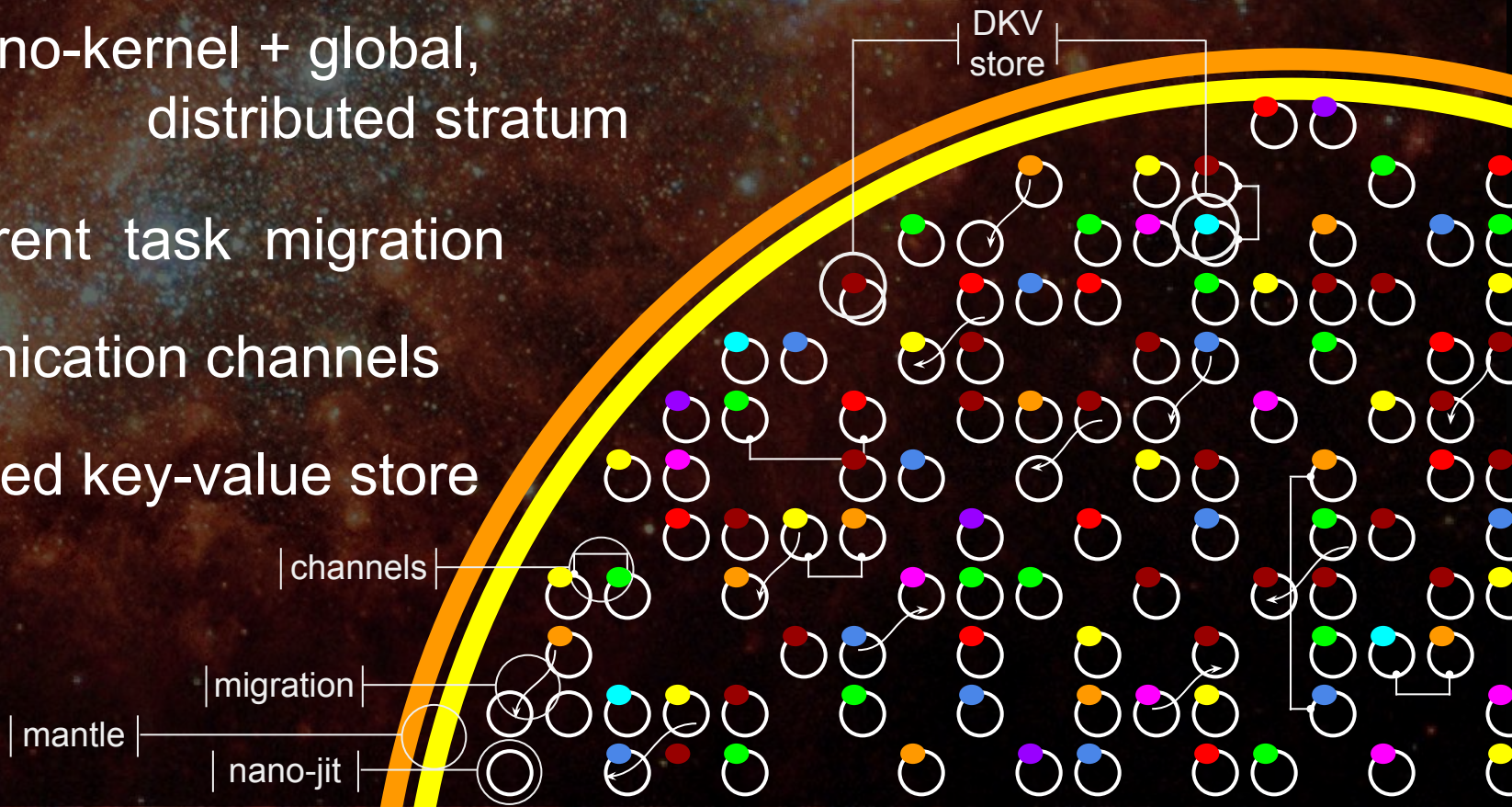
Fully-Transparent Distribution

local nano-kernel + global,
distributed stratum

transparent task migration

communication channels

distributed key-value store



Language ⇔ System

A language and RT for writing distributed apps

```
r,s = sys.async(http.crawl, [seed, 20, postprocess])
sys.status(r,s)
```

Objects (in distributed FS, interchange, serialization)

```
import "http" ==  fs.get("http") //+some checksumming and versioning
```

Functions (overriding, modularity, composition)

```
http.expose("/proc", curried(some, state)) //+ some RESTful conventions
```

Focus on fully interactive use

Unrestricted abstraction access from REPL
    Everything compiled on-the-fly

Live object introspection, module loading and function overloading

Optional Syntactic Annotations

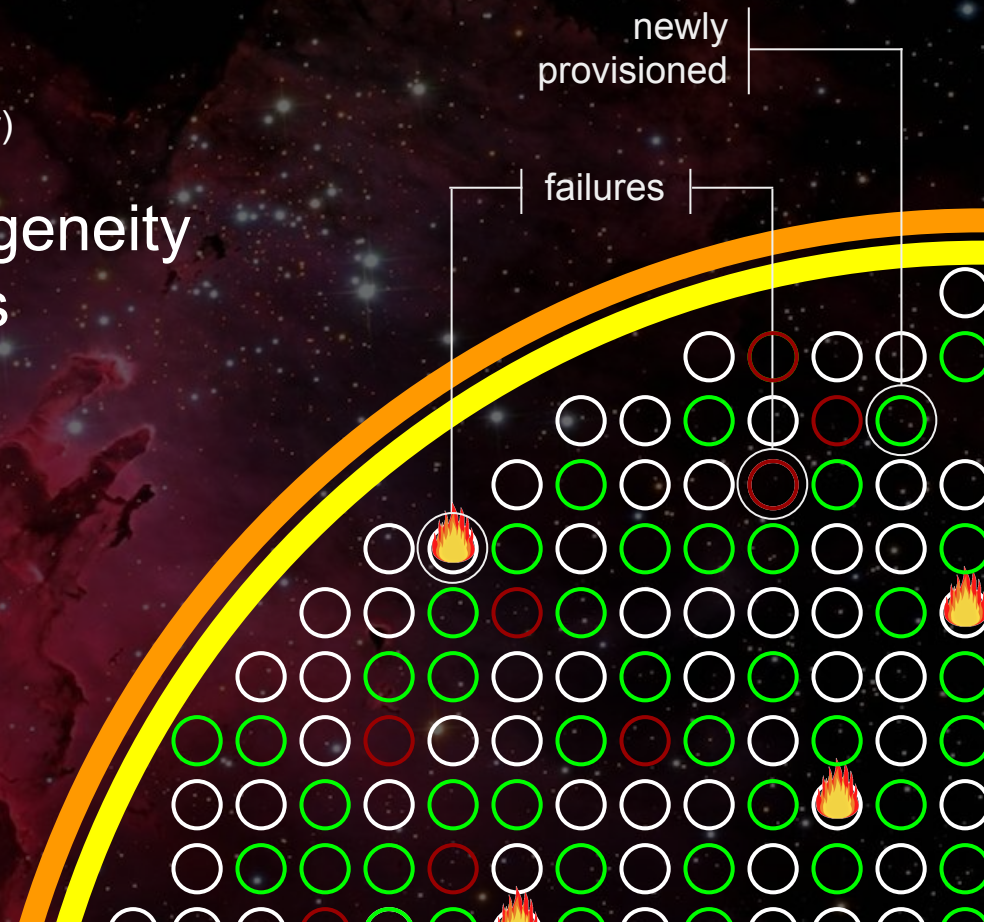
Systems that *feel* different

Blissful ignorance!

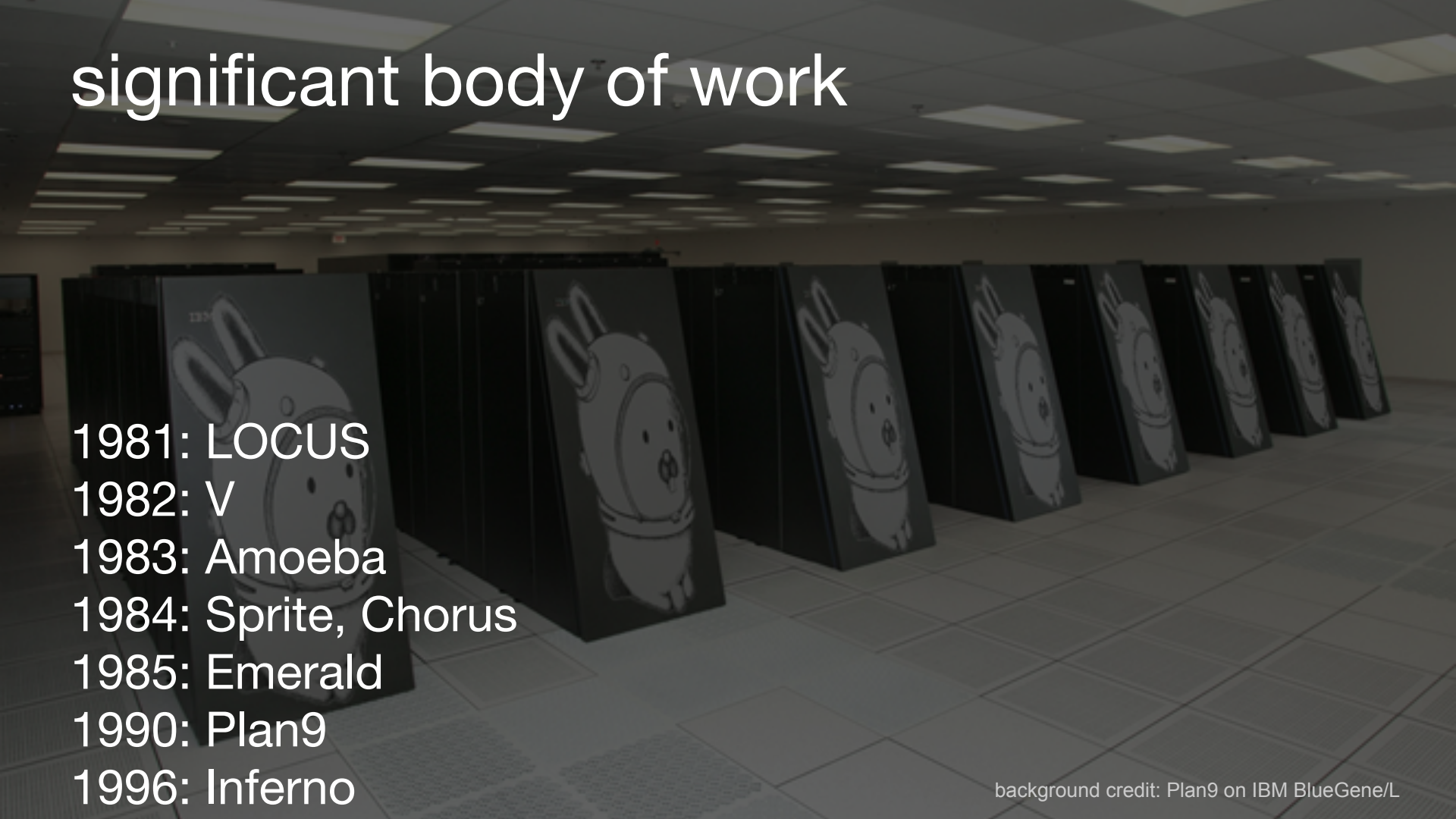
(still: tools for power users to get status -- see paper)

Dynamic Provisioning, Heterogeneity
Changing Topologies, Failures

```
m31> local sd : String = “http://ndr.md”
m31> local fn = jam.unzip
              . split
              . map(fs.incr)
m31> s,r = sys.async(url.get,
                     { seed: sd
                     , filter: mime.app.zip
                     , postprocess: fn
                     })

m31>
```



significant body of work



1981: LOCUS
1982: V
1983: Amoeba
1984: Sprite, Chorus
1985: Emerald
1990: Plan9
1996: Inferno

background credit: Plan9 on IBM BlueGene/L

similar observations from other academics and non-academics

..: Google's Borg
2011: Mesos/Spark
2012: Osprey
2013: Unikernels, Docker, Urbit
2014: DIOS, Holistic Runtime, Mesosphere, Kubernetes
2015: Meteor's "Galaxy", VMWare's "CargOS"

	UNIX	Andromeda
Influence from Existing	A Lot (Multics, coroutines, B, ..)	A Lot (DHTs, Migration, MP, ..)
HW Focus	Cheaper/lower-capability	Cheaper/lower-capability
Abstraction Axes	FS, Shell, Process, IPC, file ..	DFS, REPL, Tasks, MP, obj ..
Implementation Language	Higher-than-usual	Higher-than-usual
Workloads	Single-node	Massively-distributed
Interactive vs. “Apps”	Different (C vs. Shell)	Same Language
Isolation	Hardware	Software
Kernel	Monolithic	Nano-JIT
Environment	Multi-user	Single-user

More details in the paper (e.g., scheduling, naming, sandboxing)