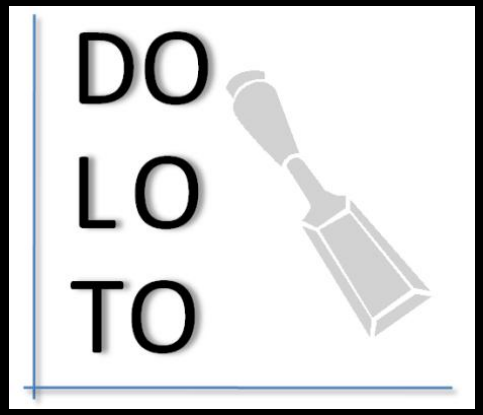


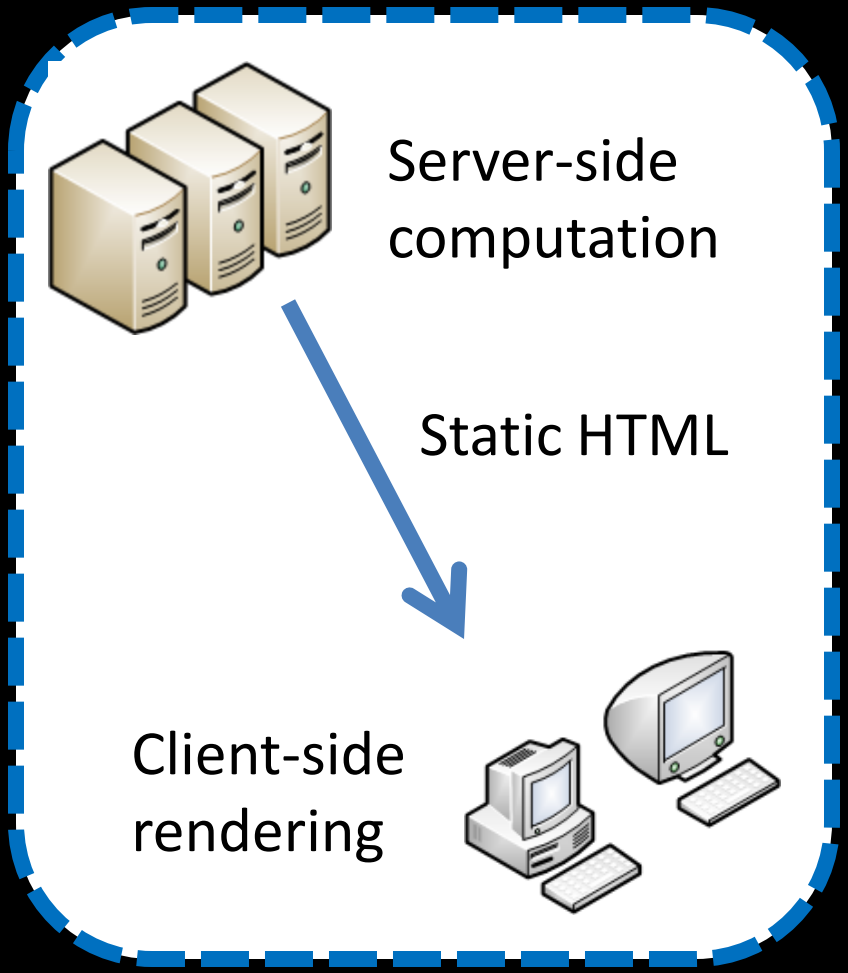
Doloto:

Code Splitting for Web 2.0 Applications

Ben Livshits and Emre Kiciman
Microsoft Research
Redmond, WA

Web 2.0 is Upon Us

Web 1.0 → Web 2.0


Server-side
computation
Static HTML
Client-side
rendering

Distributed Applications

Server

Clients

Catch-22

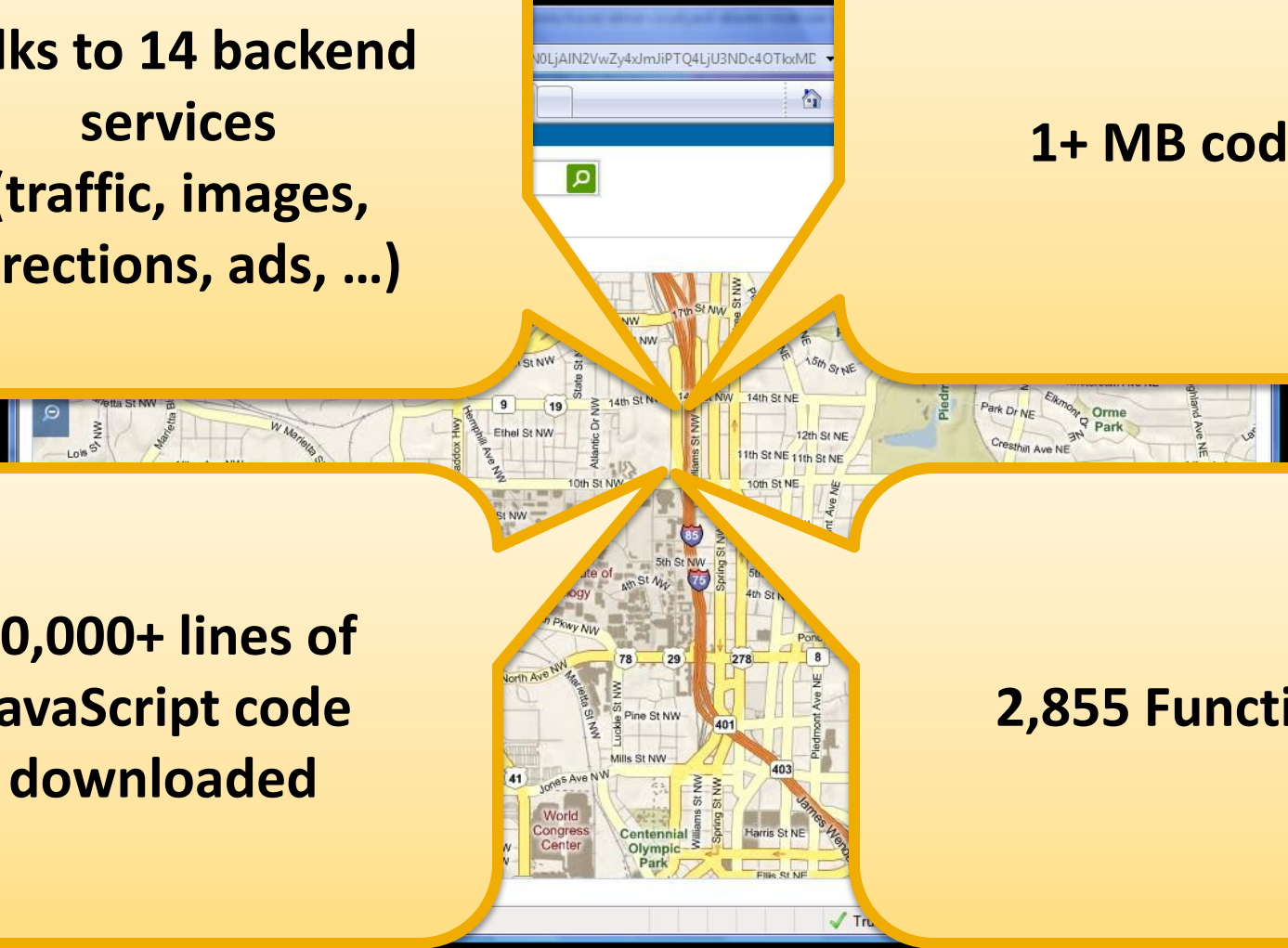
Move code to client for responsiveness

Execution can't start without the code

A Web 2.0 Application Disected

Talks to 14 backend services (traffic, images, directions, ads, …)

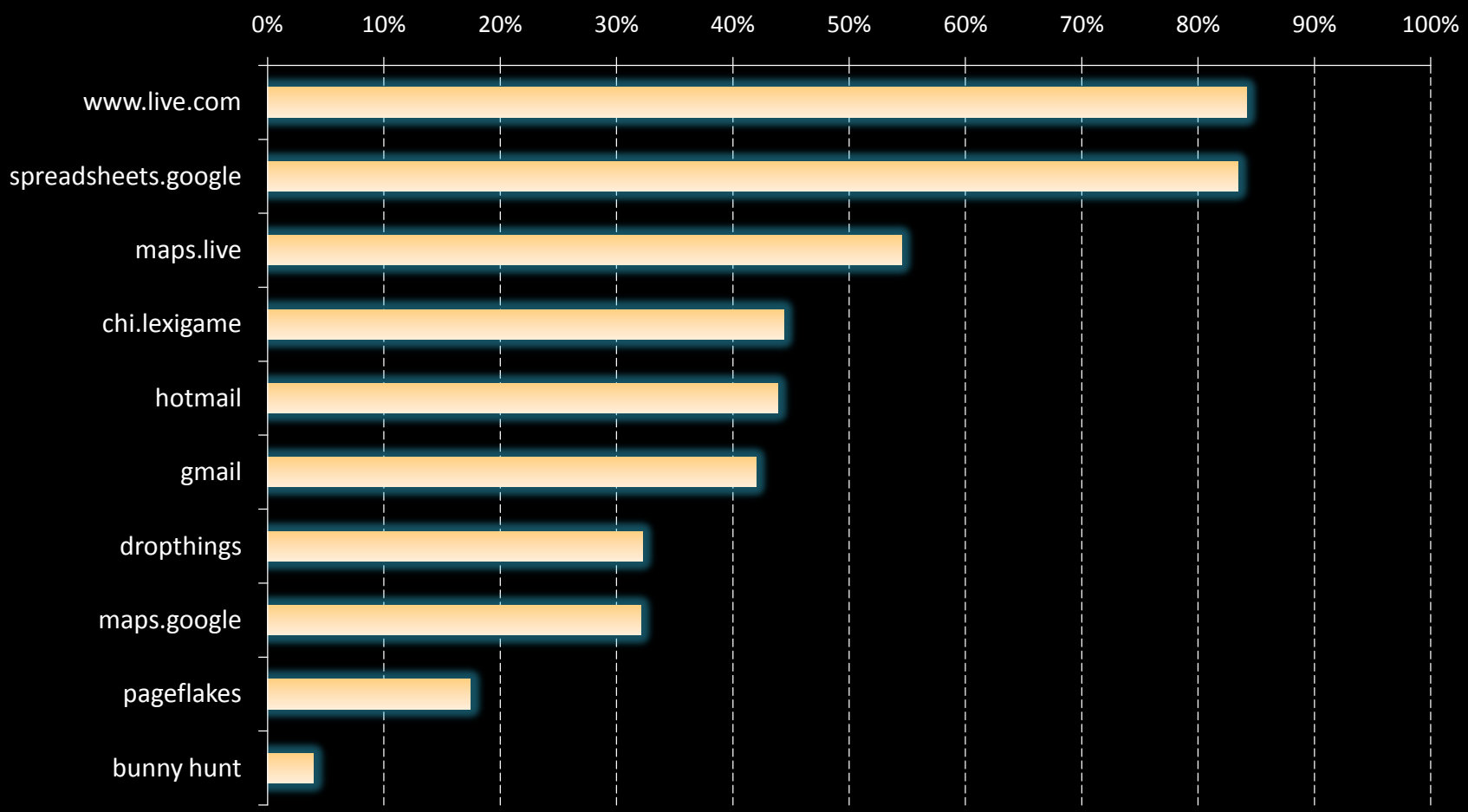
1+ MB code



70,000+ lines of JavaScript code downloaded

2,855 Functions

Lots of JavaScript Code



Motivation for Doloto

- Idea behind Doloto
  - Start with a small piece of code on the client
  - Download required code on demand (pull)
  - Send code when bandwidth available (push)
- Leads to better application responsiveness
  - Interleave code download & execution
  - Faster startup times
  - Rarely executed code is rarely downloaded

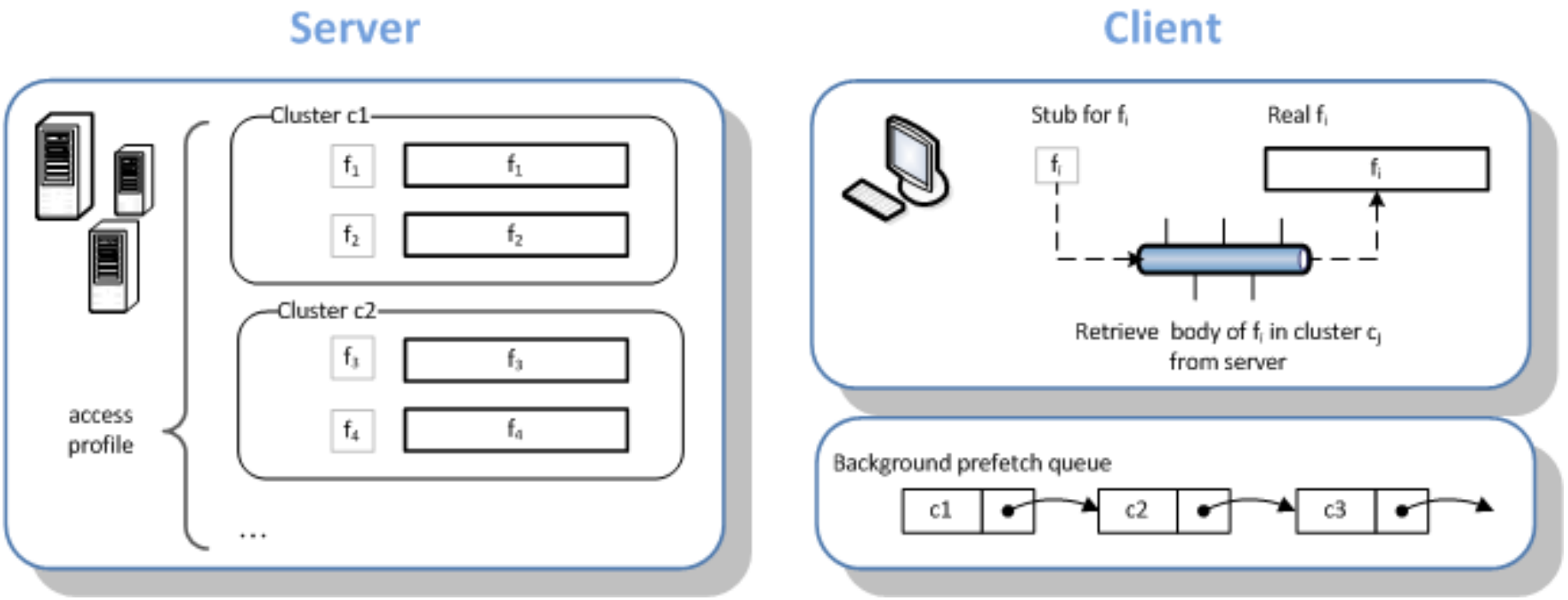
Doloto: the Steps

1. [**training**] Runtime training to collect access profile

2. [**rewriting**] Function rewriting or "stubbing" for on-demand code loading

3. [**prefetch**] Background prefetch of clusters as the application is running

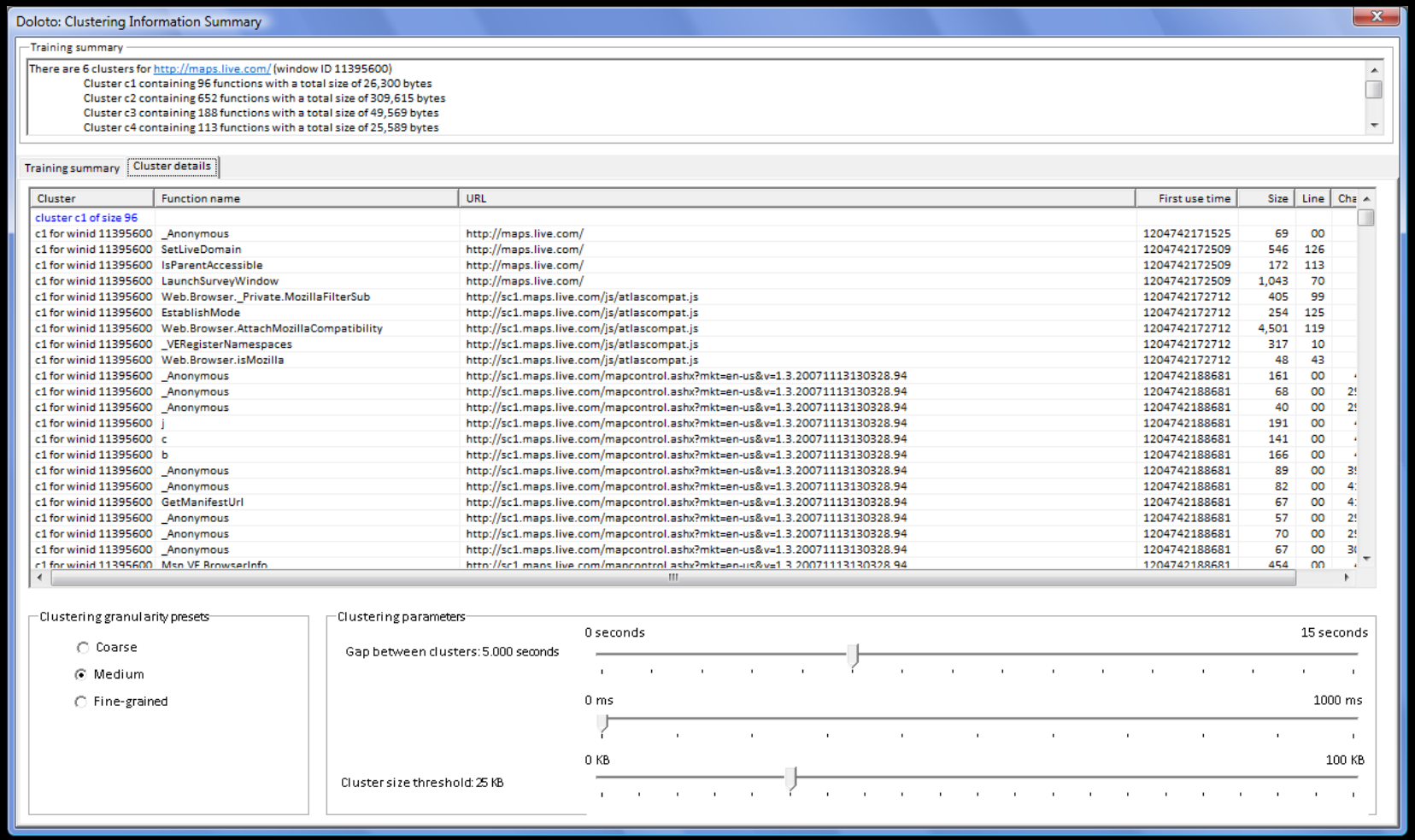
Runtime Picture



Training Phase

- Instrument every function
- Record the first-execute timestamp
- Look for gaps to find clusters

Doloto Training Tool



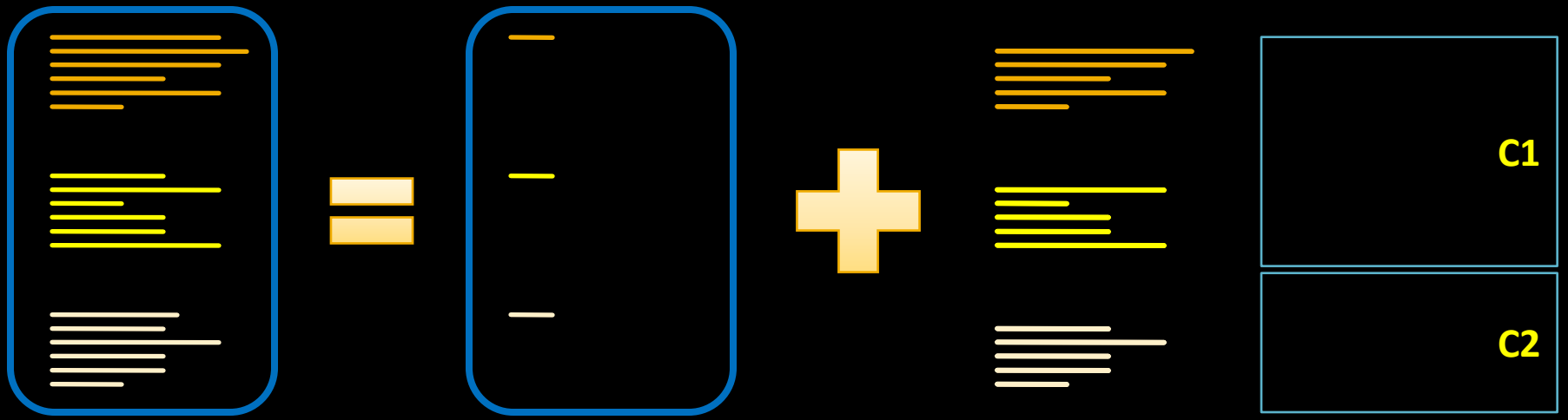
Architecture of Doloto

1. [**training**] Runtime training to collect access profiles

2. [**rewriting**] Function rewriting or "stubbing" for on-demand code loading

3. [**prefetch**] Background prefetch of clusters as the application is running

Code Rewriting



- Rewrite JavaScript code one file at a time
- Recombine clusters into individual files

Automated Function Splitting

```
var g = 10;
function f1(){
    var x=g+1;
    …
    …
    …
    …
    …
    return …;
}
```

```
var g = 10;

var real_f1;
function f1() {
    if(!real_f1){
        var code = load("f1");
        real_f1 = eval(code);
        f1 = real_f1;
    }
    return real_f1.apply(this,
                         arguments);
}
```

```
eval($exp("f1"), "");          // 21 chars
```

Architecture of Doloto

1. [**training**] Runtime training to collect access profiles

2. [**rewriting**] Function rewriting or "stubbing" for on-demand code loading

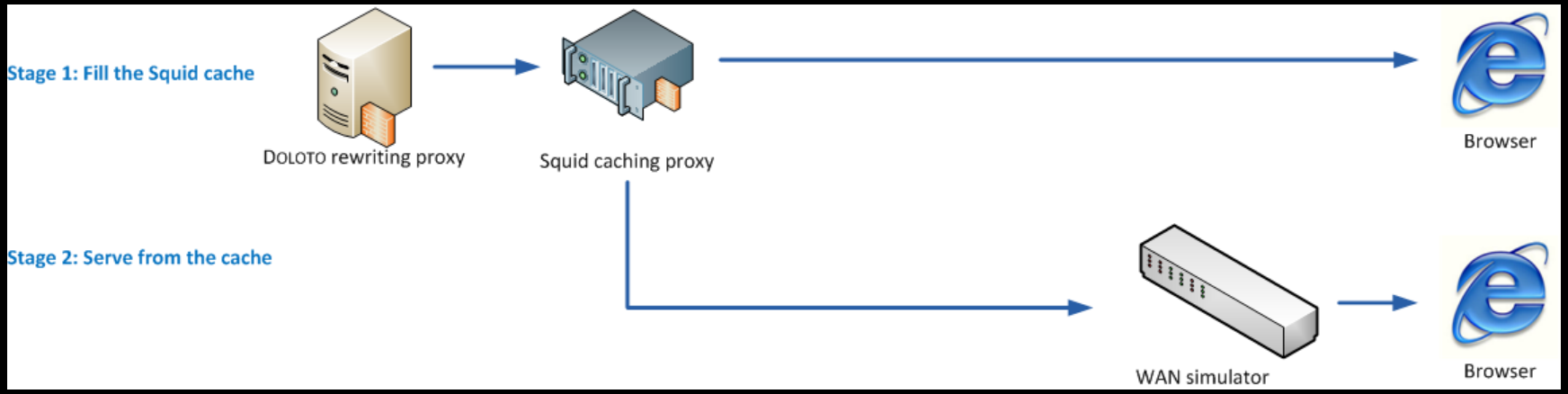
3. [**prefetch**] Background prefetch of clusters as the application is running

Cluster Prefetching

```
1   var xhr = new XmlHttpRequest();
2   function next_cluster(){
3       xhr.open("http://code.server.com/next",
4       /* asynchronous AJAX call */     true);
5
6       xhr.onreadstatechange = handle_cluster;
7       xhr.send(null);
8   }
9
10  function handle_cluster(){
11      if (xhr.readyState != 4)  { return; }
12      var code = xhr.responseText;
13      if (code == "") return; // last cluster
14
15      // split code into function bodies
16      foreach(<func_name, func_code> in code) {
17          func[func_name] = func_code;
18      }
19
20      // go fetch the next cluster
21      next_cluster();
22  }
23
24  // initial invocation of next_cluster
25  // after the document is done loading
26  document.attachEvent("onload", next_cluster);
```

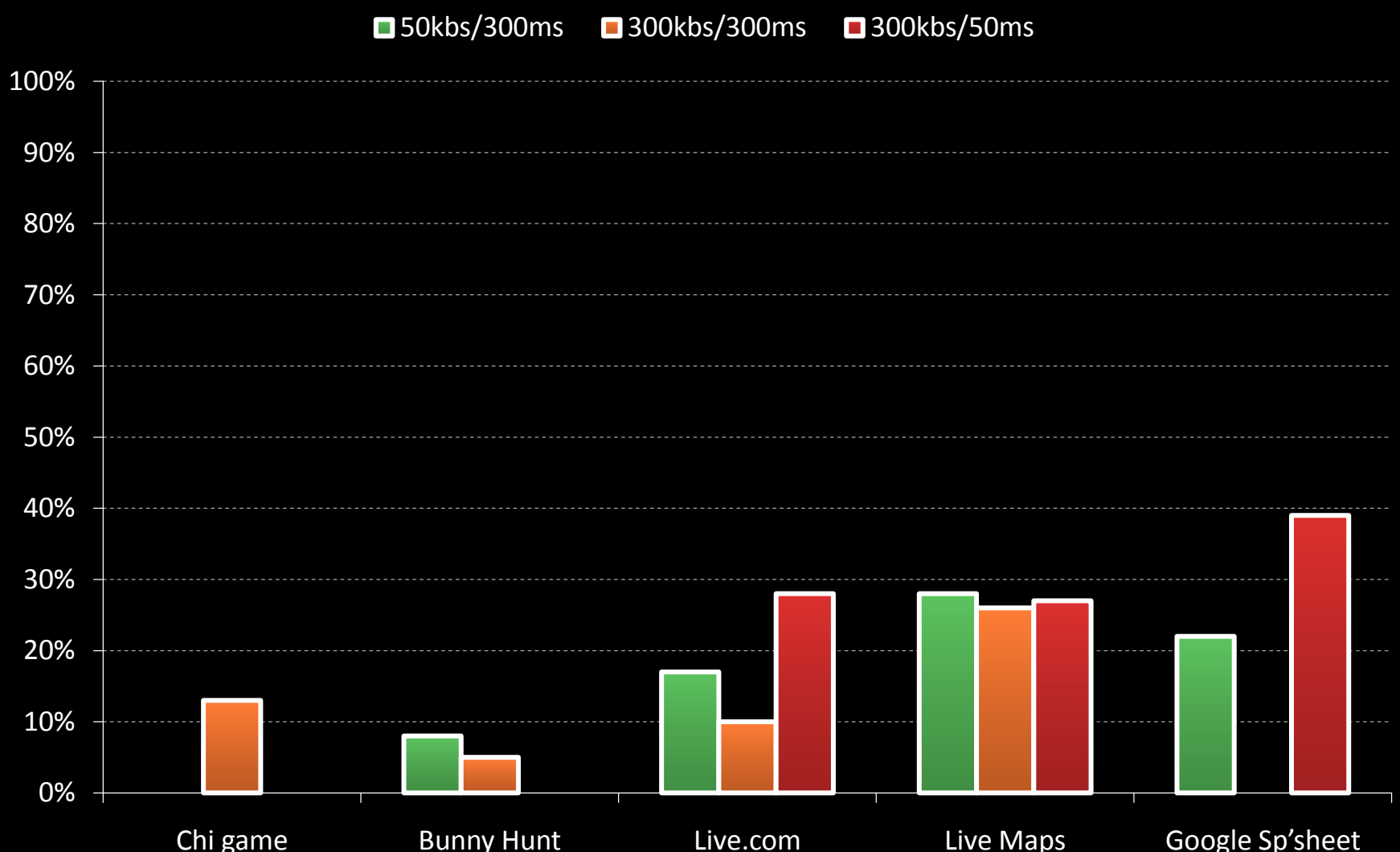
Application Benchmarks

Application	Download Size
Chi game	104
Bunny Hunt	16
Live.com	1,436
Live Maps	1,909
Google Spreadsheets	499



Size Savings with Doloto

Runtime Savings with Doloto



Conclusion

- Doloto: effective profile-driven optimization
- Our approach is general: Silverlight
- Enables larger more complex distributed apps
- Dynamic code loading for distributed applications of the future

Contact us

Ben Livshits (livshits@microsoft.com)

Doloto MSR _