Reinventing MMOs

A

"antemortem"

Raph Koster & Sean Riley

President *Lead Programmer*

areae

Why reinvent MMOs?

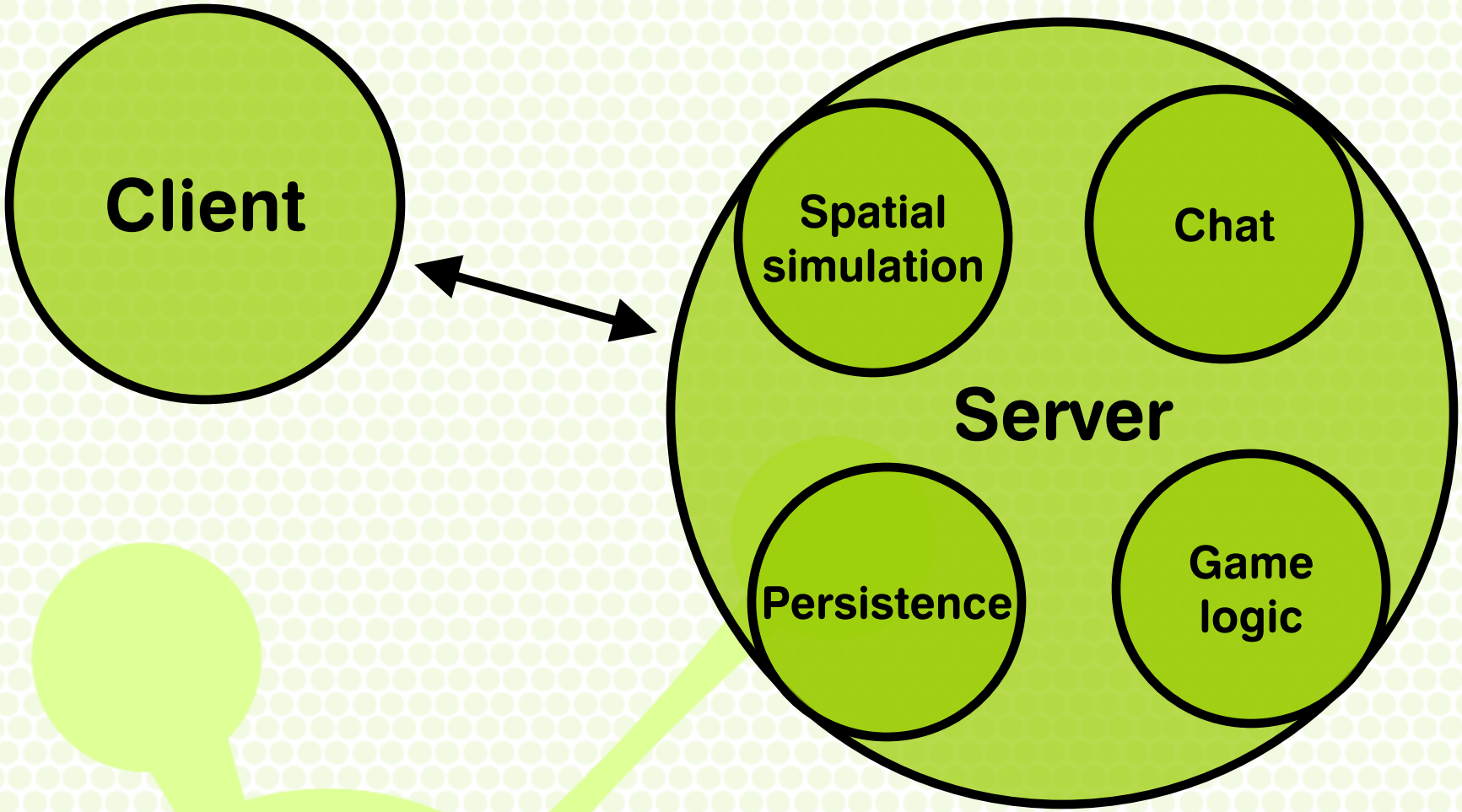
- They are just too hard to make now
  - *Each one is a custom-crafted moon shoot.*
- Poorly integrated with rest of Internet
- Which leads to
  - *Cost: only the big kids can play*
  - *Cancellations: get it perfect or go home*
  - *Wheel reinvention: chat system #207*
  - *Lack of innovation: "make it like WoW…"*

How MMOs work today

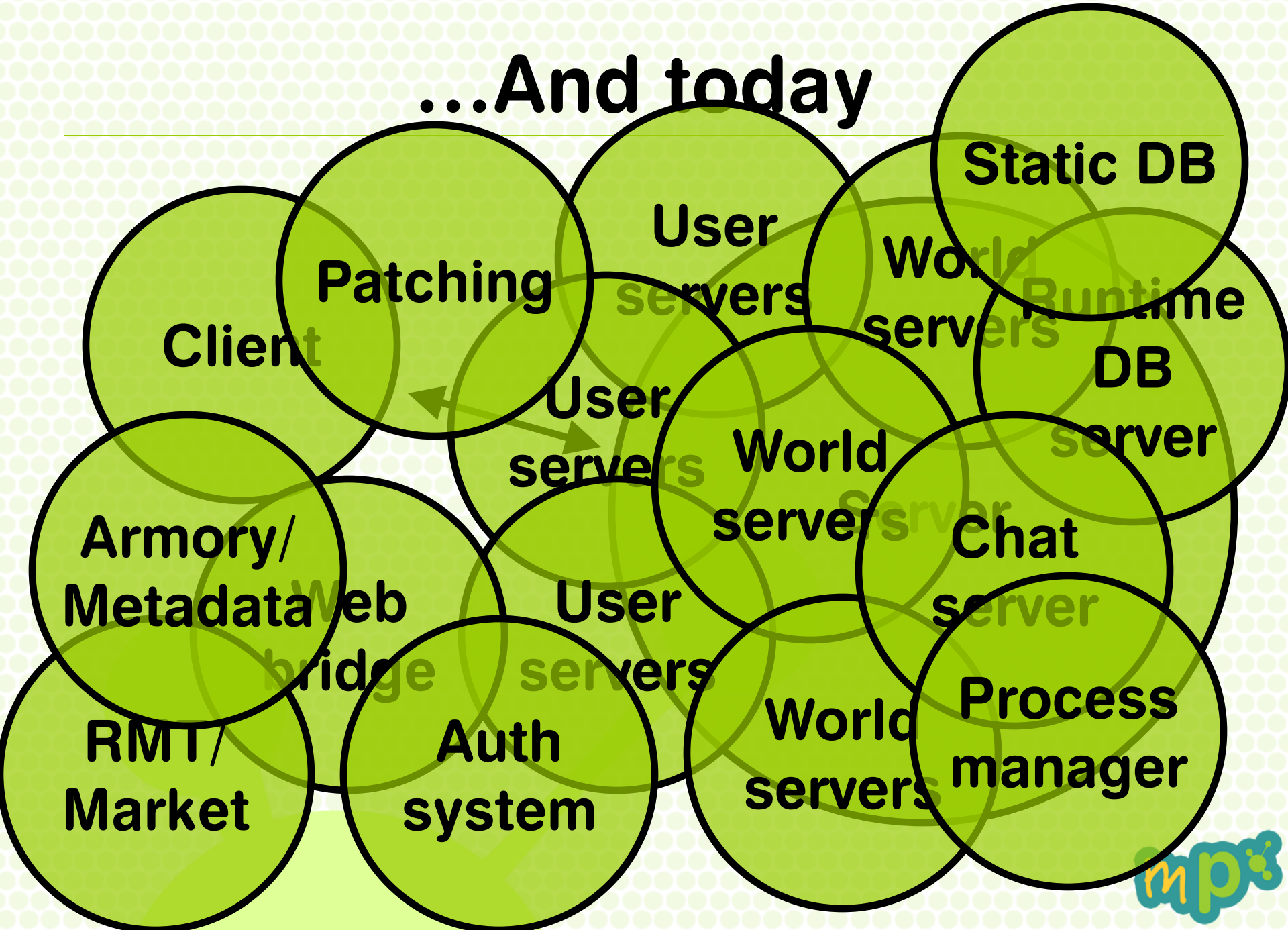
- Monolithic, giant servers
- All services contained within the server
- Complex server cluster architectures
- Tight dependency between client and server

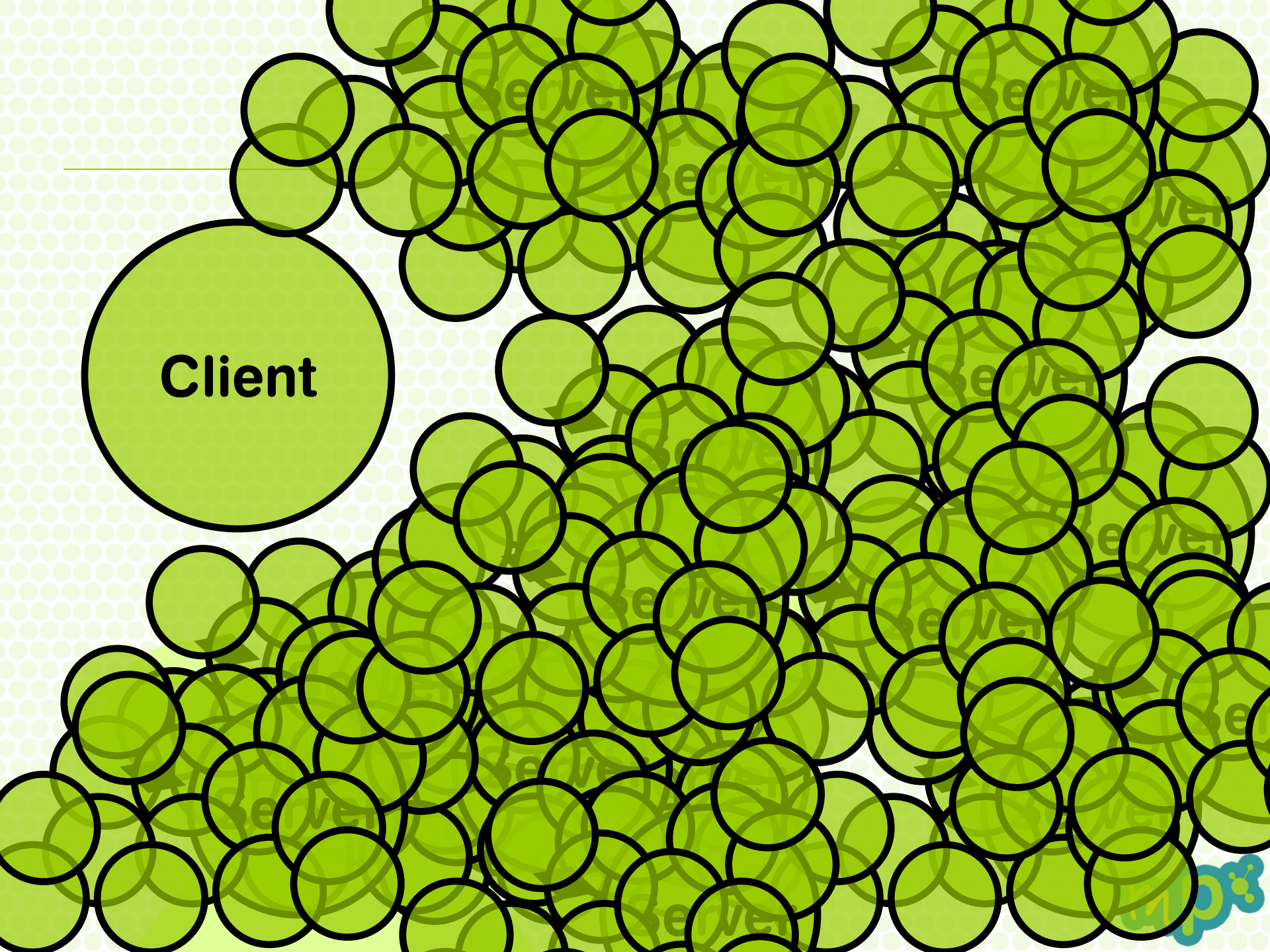
There are virtually no successful re-uses of MMO servers

How it once was



…And today


Static DB
Patching
Client
User servers
World servers
Runtime
DB server
User servers
World server
Chat server
Armory/ Metadata
Web bridge
User servers
Auth system
World servers
Process manager
RMT/ Market


Client

The alternative

- There exists a system which is
  - *Low cost to develop for*
  - *Extremely scalable*
  - *Robust and pretty future-proof*
  - *Highly modular and reusable*
  - *Has resulted in gigantic explosions of creativity at both pro and amateur levels*

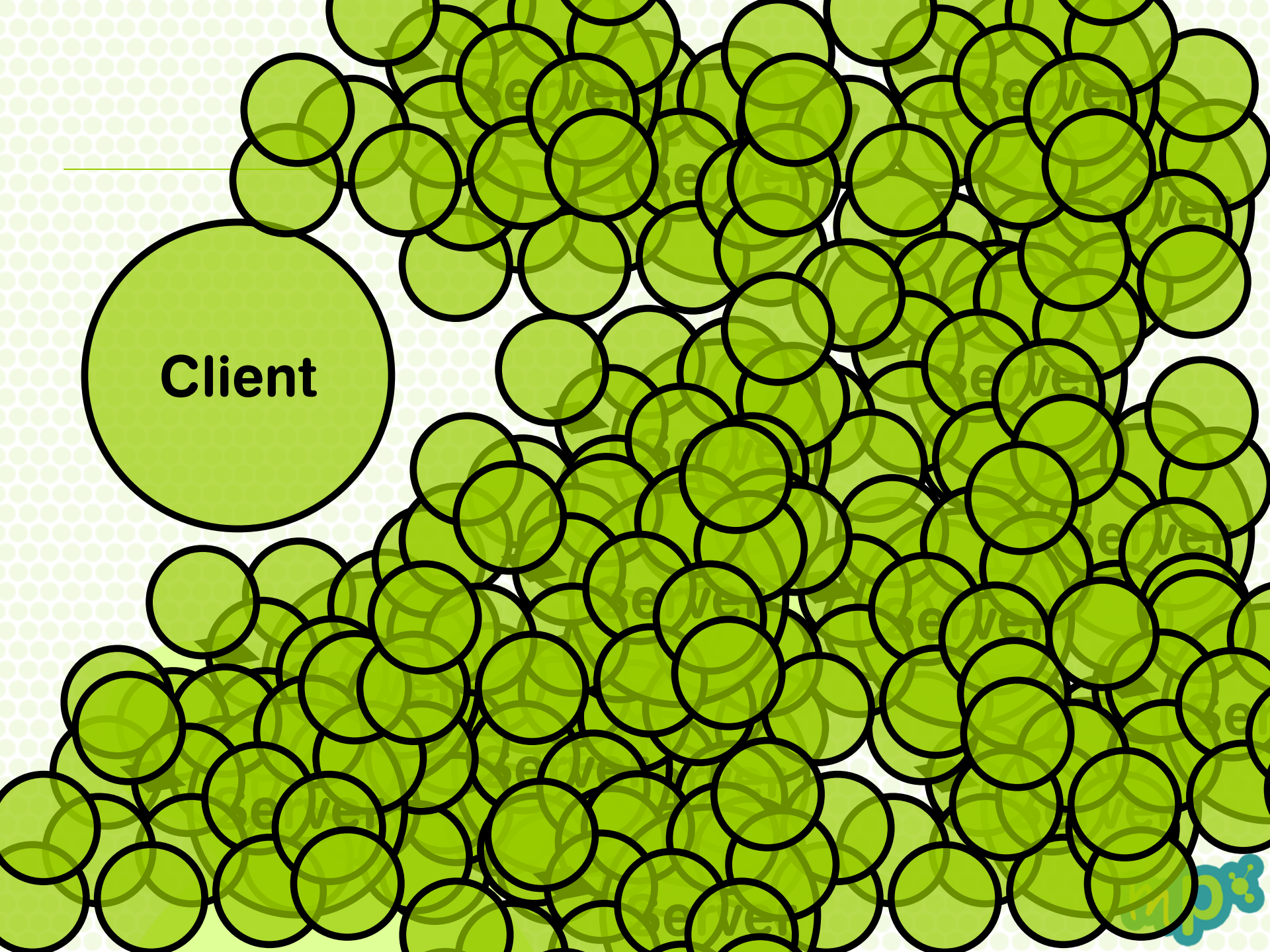
The web.

Why does the Web work today?

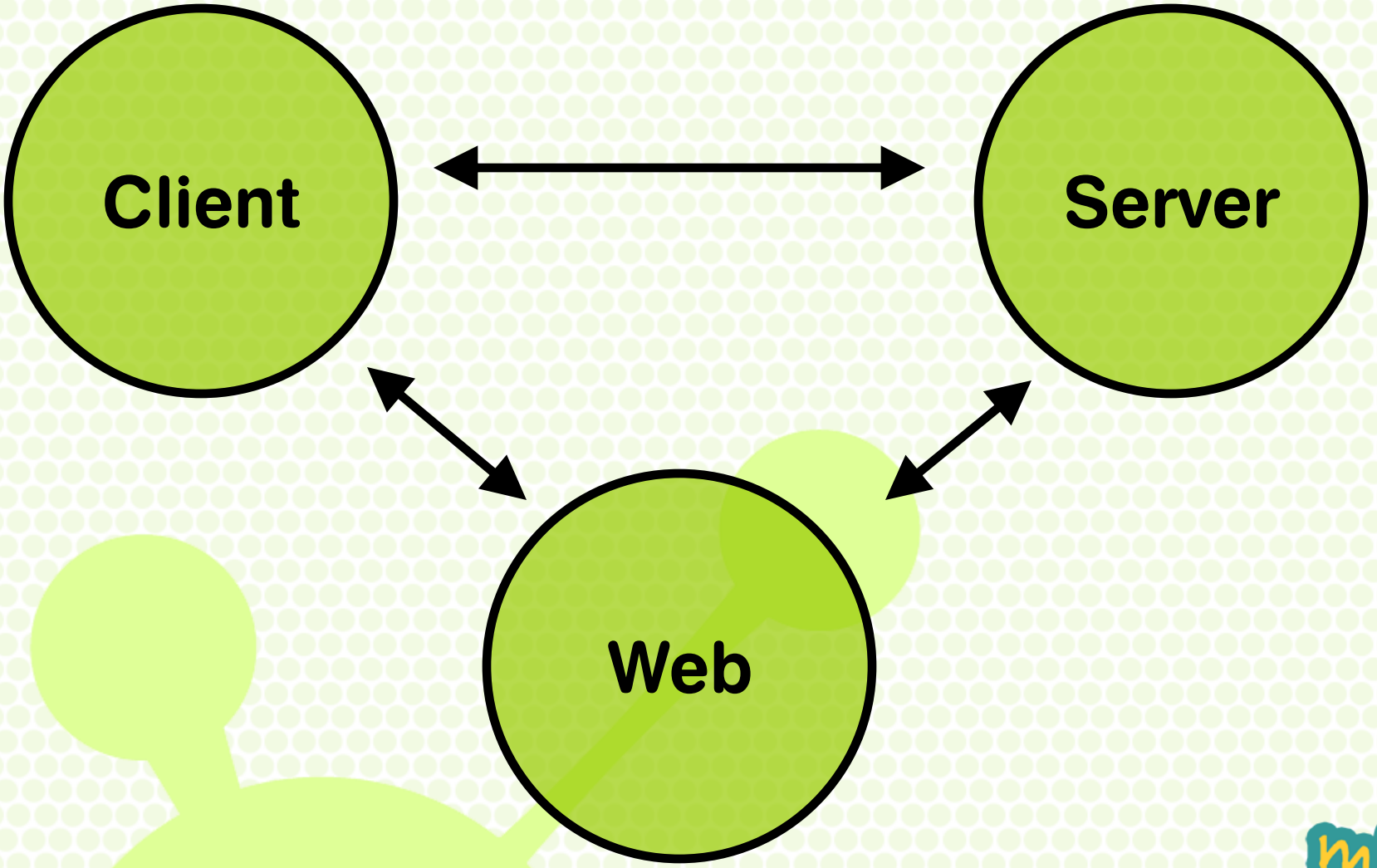
- HTML
- Mozilla
- Apache
- CGI
- CSS
- DNS
- Google

An MMO that looks like the web

HTML	*Open markup language*
Mozilla	*Reference browser*
Apache	*Assumption-free server*
CGI	*Scripted behaviors*
CSS	*Templated content*
DNS	*Distributed, segmented routing*
Google	*Rich in metadata*


Client

Server graph – MP



A philosophical difference

Put virtual worlds on the web,

not the web in a virtual world

- Add synchronous realtime multiuser capabilities to the web
  - *Not REST*
  - *Leverage tech from game industry that web doesn't know how to do*

Demo

How Metaplace was built

- Iterate and prototype!
- Decouple components!
- Don't optimize!
- Use what already exists!
- Start from the ridiculously simple!
  - *BASIC, flat text file, TCP socket, pipe-delimited, and see how far you can go*

"HTML"

- Started handcoding tags in a text file
  - *Rigid tag structure: 10 bytes for tag*
  - *Split by pipe*
- Multiple formats for tags
  - *GML, JSON, binary, etc*
- XML schema for protocol
- Protocol negotiation

Lesson: Verbose/inefficient protocols, negotiate up to efficiency

"Mozilla"

- 1st display client – BASIC!
  - *2d overhead only at first*
  - *HTTP for all assets*
- 2nd display client – Flash
- 3rd client C++
- 4th client Flash again
- 5th, just starting – standardized XML schema-based setup

Lesson: Clients are stupid.

“Apache”

- 1st server was a text mud codebase
  - *Synchronous, basic spatial sim*
- 2nd server Twisted Python
  - *Our own stripped-down obj structure*
- 3rd server C++
  - *Platform services: spatial sim, physics, persistence, auth*
- Server gains web server
  - *Objects get URLs*
- Server gains web browser
  - *All platform services which can be moved* are *moved*

Lesson: Server is as light as possible: a thin aggregator

“CGI”

- At first, had no behaviors at all.
- Hardcoded behaviors in Python server
- Lua backend on Python server
- Swap out Python server for C++
  - *Inline CGI is Lua*
  - *Out of process CGI is web services*
- Future: load scripts from URLs

Lesson: Behavior is small, atomic, componentized, not monolithic

“CSS”

- Flat files first
  - *from text file, then from web service*
- Real time increments via tags
- Keep web service, backend to DB
  - *Client-side caching*
- XML service added
- Packaging subsets of stylesheet

Lesson: Separate display from behavior; clients just render.

“DNS”

- Hardcoded single server
- Distributed servers, manual routing
- “Worldlist” true distributed system
- Worldlist too big, change to WNS
  - *Web exposure of server data, WNS*
  - *Web service for WNS*
- Web services for all sorts of metadata

Lesson: Don’t cram the Internet into your platform

"Google"

- PHP for worlds & stylesheet data
- Then two pieces: portal, stylesheet
  - *Built website on top of portal data*
  - *Built web services on stylesheet DB*
- Added Web 2.0 features to portal stuff
  - *ID, ratings, reviews, community, presence*
- Built web services to expose portal to everywhere

Lesson: it's no good if you cannot find stuff/friends

What Went Right

- Use web solution when available!
- KISS! Don't design for edge cases.
- Tags! Our network stream never breaks.
- Way picky hiring
  - *Especially, the hybrid team, half web half game*
- Multiclient early: forced us to be honest
- *Don't optimize!*
  - » Michael Jackson's rules of optimization from *Principles of Program Design (Academic Press, 1975)*

Recapitulate history with incremental platform capabilities

What Went Wrong

- Skeletal animation system.
  - *Prototype quick – actual thing never finished.*
- "Simple" in game world versus "simple" in web world – not the same!
- Needed the RPG, it's what customers wanted.
- Content people earlier, push the pipelines.
- We fell off the multiclient bandwagon.
- Not being task-based in tool creation

www.metaplace.com