Lua

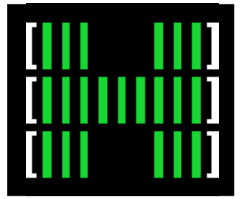
an embeddable, high-performance scripting language and its applications

Hisham Muhammad
hisham@inf.puc-rio.br

PUC-Rio, Rio de Janeiro, Brazil

Introductions

- Hisham Muhammad
- PUC-Rio
  - University in Rio de Janeiro, Brazil
- LabLua research laboratory
  - founded by Roberto Ierusalimschy, Lua's chief architect
- lead developer of LuaRocks
  - Lua's package manager
- other open source projects:
  - GoboLinux, htop process monitor



What we will cover today

- The Lua programming language
  - what's cool about it
  - how to make good uses of it
- Real-world case study
  - an M2M gateway and energy analytics system
  - making a production system highly adaptable
- Other high-profile uses of Lua
  - from Adobe and Angry Birds
    to World of Warcraft and Wikipedia

Lua?

- ...is what we tend to call a "scripting language"
  - dynamically-typed, bytecode-compiled, garbage-collected
  - like Perl, Python, PHP, Ruby, JavaScript...
- What sets Lua apart?
  - Extremely portable: pure ANSI C
  - Very small: embeddable, about 180 kiB
  - Great for both embedded systems and for **embedding into applications**

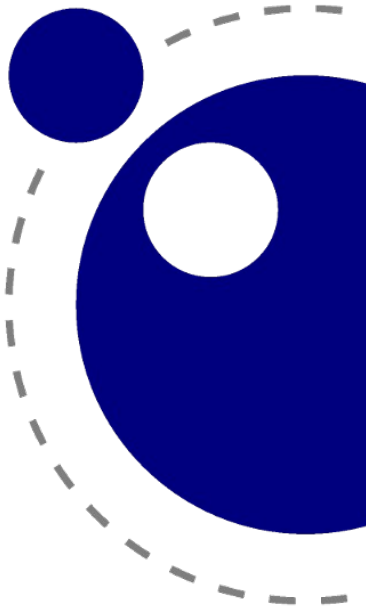
Lua is fully featured

- All you expect from the core of a modern language
  - First-class functions (proper closures with lexical scoping)
  - Coroutines for concurrency management (also called "fibers" elsewhere)
  - Meta-programming mechanisms
    - object-oriented
    - functional programming
    - procedural, "quick scripts"

To get licensing out of the way

- MIT License
- You are free to use it anywhere
- Free software projects
  - OSI certified license
- Proprietary commercial projects
  - No royalties

What Lua looks like

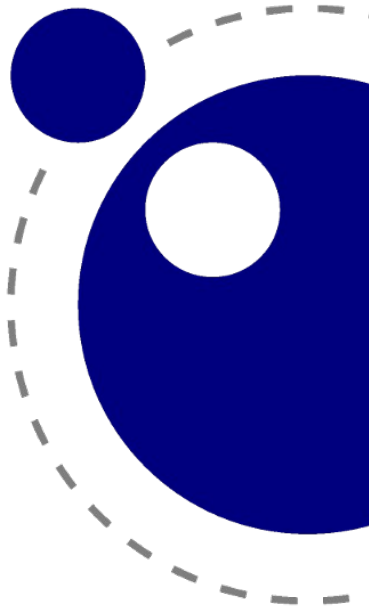


```lua
function process(filename, fn, ...)
    local f = io.open(filename)
    local rets = {}
    for line in f:lines() do
        rets[#rets+1] = { fn(line, …) }
    end
    f:close()
    return rets
end

matches = process("file.txt", string.find, "foo")
for i, match in ipairs(matches) do
    print(i, table.concat(match), ", ")
end
```

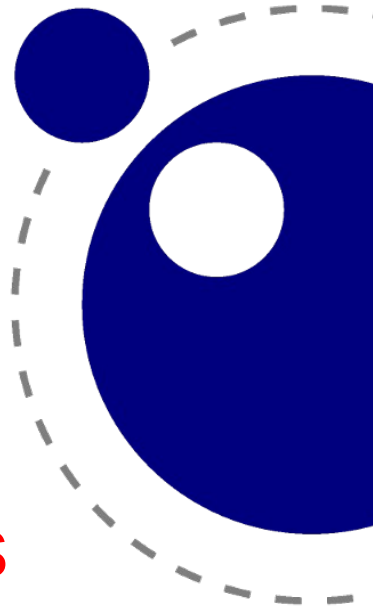
What Lua looks like

```
function process(filename, fn, ...)
    local f = io.open(filename)
    local rets = {}
    for line in f:lines() do
        rets[#rets+1] = { fn(line, …) }
    end
    f:close()
    return rets
end

matches = process("file.txt", string.find, "foo")
for i, match in ipairs(matches) do
    print(i, table.concat(match), ", ")
end
```

objects



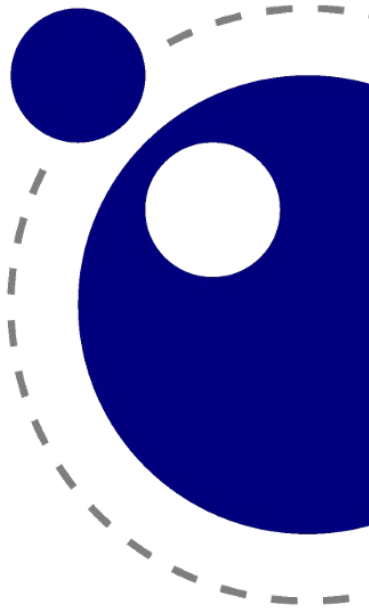
What Lua looks like



```lua
function process(filename, fn, ...)
    local f = io.open(filename)
    local rets = {}
    for line in f:lines() do
        rets[#rets+1] = { fn(line, …) }
    end
    f:close()
    return rets
end

matches = process("file.txt", string.find, "foo")
for i, match in ipairs(matches) do
    print(i, table.concat(match), ", ")
end
```

first-class functions

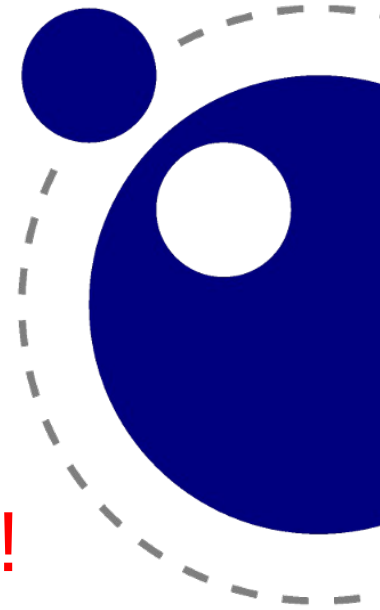
What Lua looks like



```lua
function process(filename, fn, ...)
    local f = io.open(filename)
    local rets = {}
    for line in f:lines() do
        rets[#rets+1] = { fn(line, …) }
    end
    f:close()
    return rets
end

matches = process("file.txt", string.find, "foo")
for i, match in ipairs(matches) do
    print(i, table.concat(match), ", ")
end
```

iterators

What Lua looks like



```lua
function process(filename, fn, ...)
    local f = io.open(filename)
    local rets = {}
    for line in f:lines() do
        rets[#rets+1] = { fn(line, …) }
    end
    f:close()
    return rets
end

matches = process("file.txt", string.find, "foo")
for i, match in ipairs(matches) do
    print(i, table.concat(match), ", ")
end
```

tables, tables everywhere!

So, what's the trick?

- How can it be so small? Python takes 72 MiB!

So, what's the trick?

- How can it be so small? Python takes 72 MiB!
- The answer:
  "Batteries not included!"
- The core gives you only the core: standard types and ANSI C facilities (such as files)
- Zero bloat
  - you "pay" for what you need

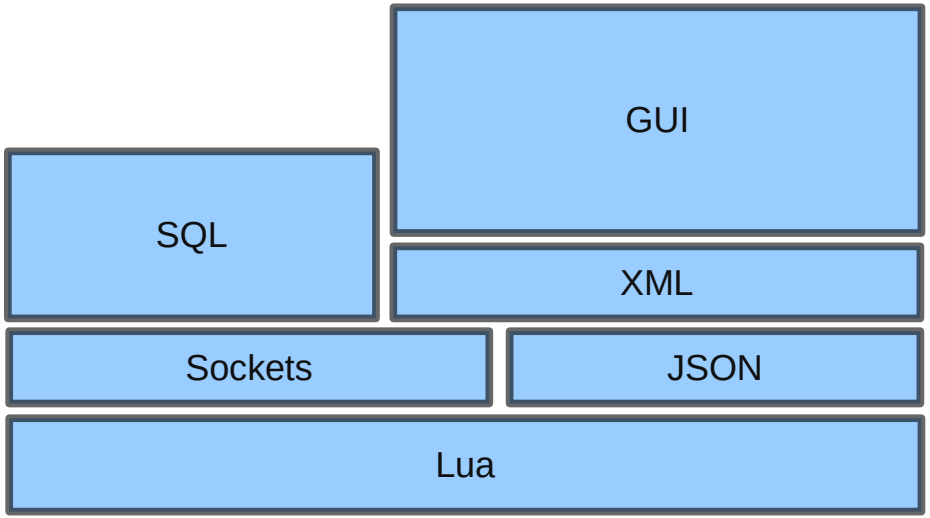
PUC RIO

For anything else: modules

- Everything else is provided through external libraries
- What do I mean by everything?

For anything else: modules

- Everything else is provided through external libraries
- What do I mean by everything? Everything:



But you don't have to implement it

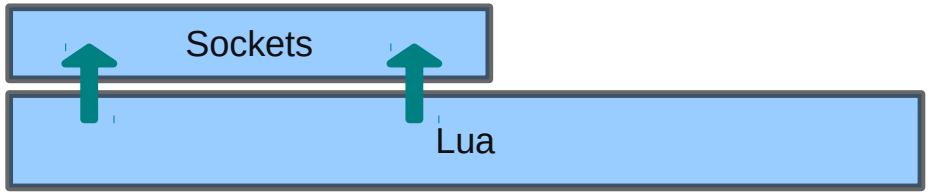
- All these libraries are already available
- It's easy to load them to your Lua environment
- LuaRocks, the package management system for Lua
  - like RubyGems, CPAN, npm, etc.
- Need sockets?

```
luarocks install luasocket
```

- (this is a shameless plug:
  I maintain LuaRocks :-))

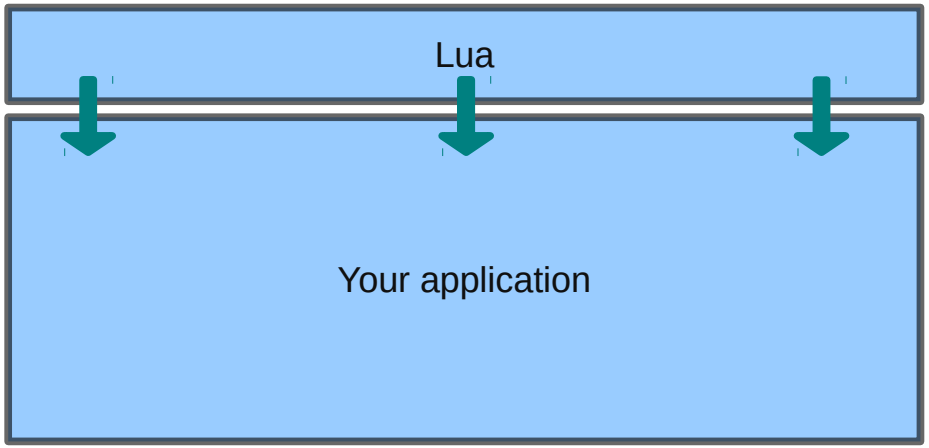
The size of your needs

- You only "pay" in memory and space for what you need
- Good for security audits



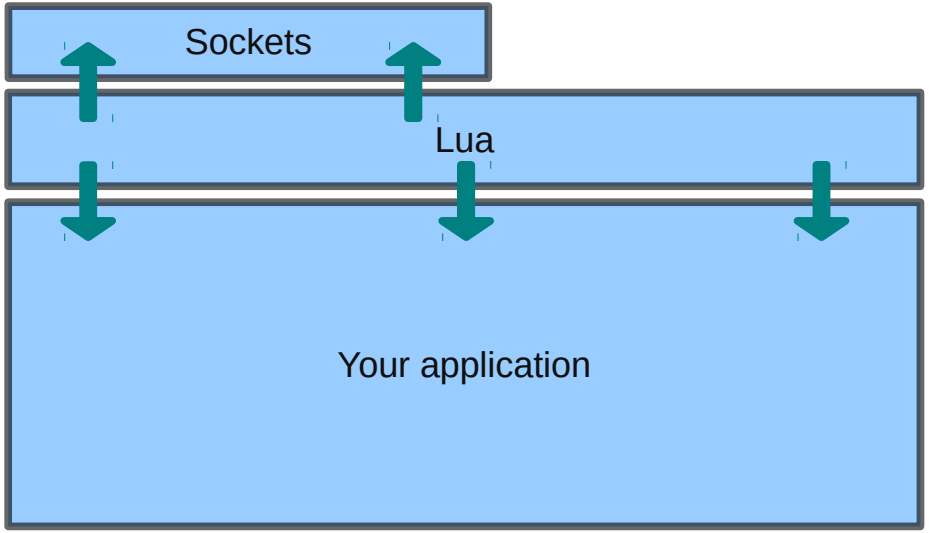
The size of your needs

- You only "pay" in memory and space for what you need
- Good for security audits
- Good for embedding in applications:



The size of your needs

- You only "pay" in memory and space for what you need
- Good for security audits
- Good for embedding in applications:



A case study: M2M gateway application

- Let's have a more concrete idea through a real-world example
- A project I worked on for a Brazilian company

ENERGIA | GERADORES | AMBIENTAL | CONECTA | CONSULTORIA

Tenha total controle sobre seu negócio a qualquer hora e em qualquer lugar!
Você não precisa perder tempo buscando tecnologias que não contemplam as suas necessidades ou gastar além do seu orçamento para ter soluções eficientes. Com anos de investimento em desenvolvimento, o Iplenix criou a ferramenta mais completa para você monitorar seus equipamentos à distância.

O Iplenix oferece soluções de alto padrão, simplificando a gestão do seu negócio e reduzindo o tempo e o custo da captação, transmissão e monitoramento de dados gerados eletronicamente, imprimindo agilidade e segurança nos seus processos, e com o melhor custo benefício do mercado.
Contrate as soluções Iplenix e insira sua empresa na era da Internet e do acompanhamento online, melhorando seus resultados.

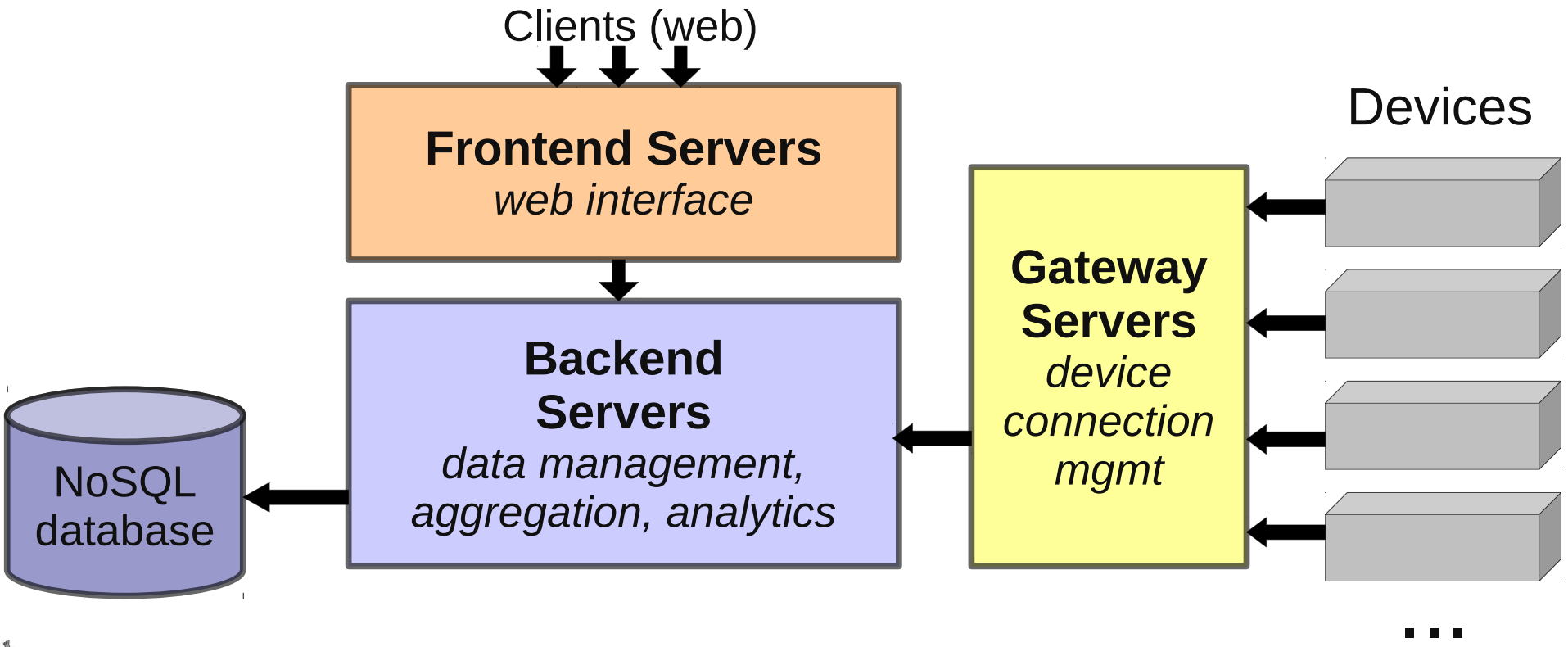
Acesse aqui seu serviço



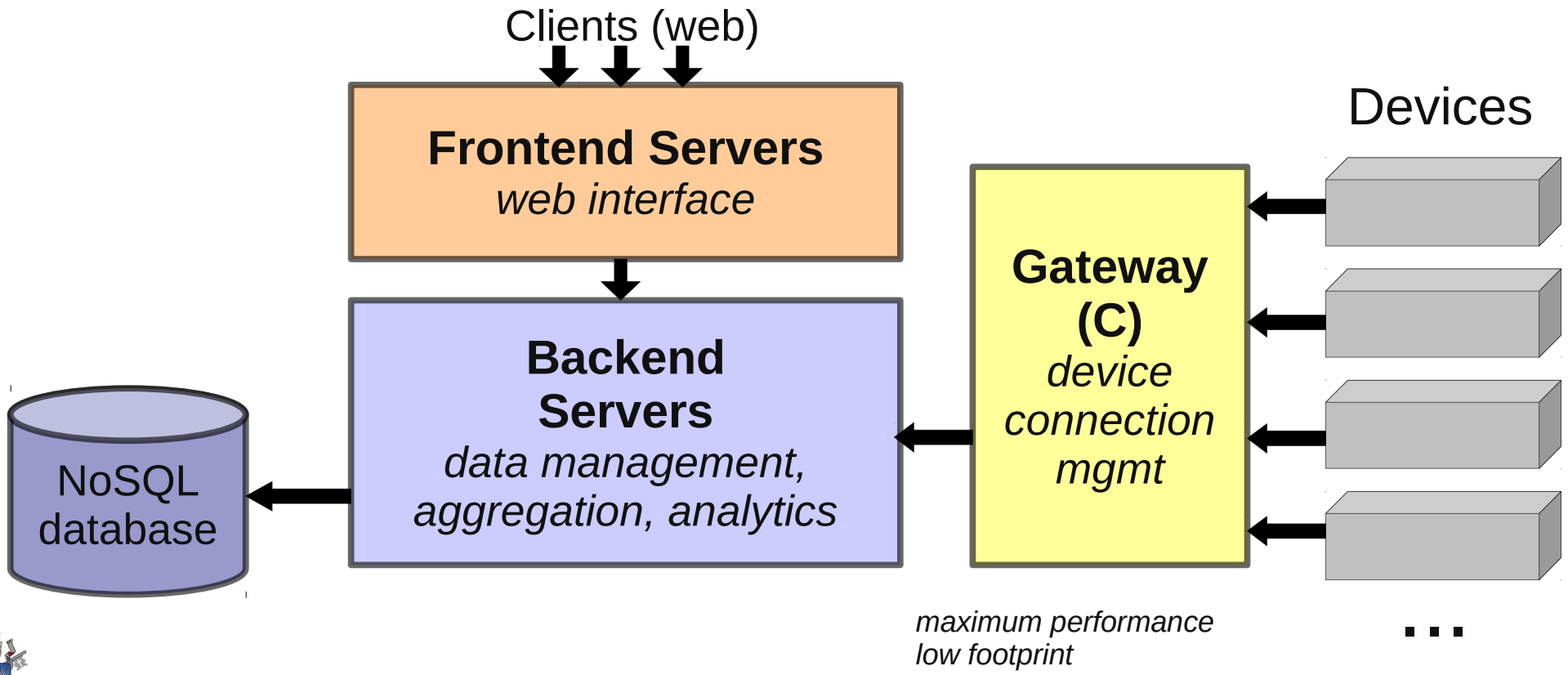

The context

- Telemetry for energy equipment
  - Power generators, consumption meters
  - Data collection and statistics
- Analytics on energy bills
- Machine-to-machine (M2M) communication:
  - Get custom hardware (legacy ports, etc.) in the net
  - Aggregate their data

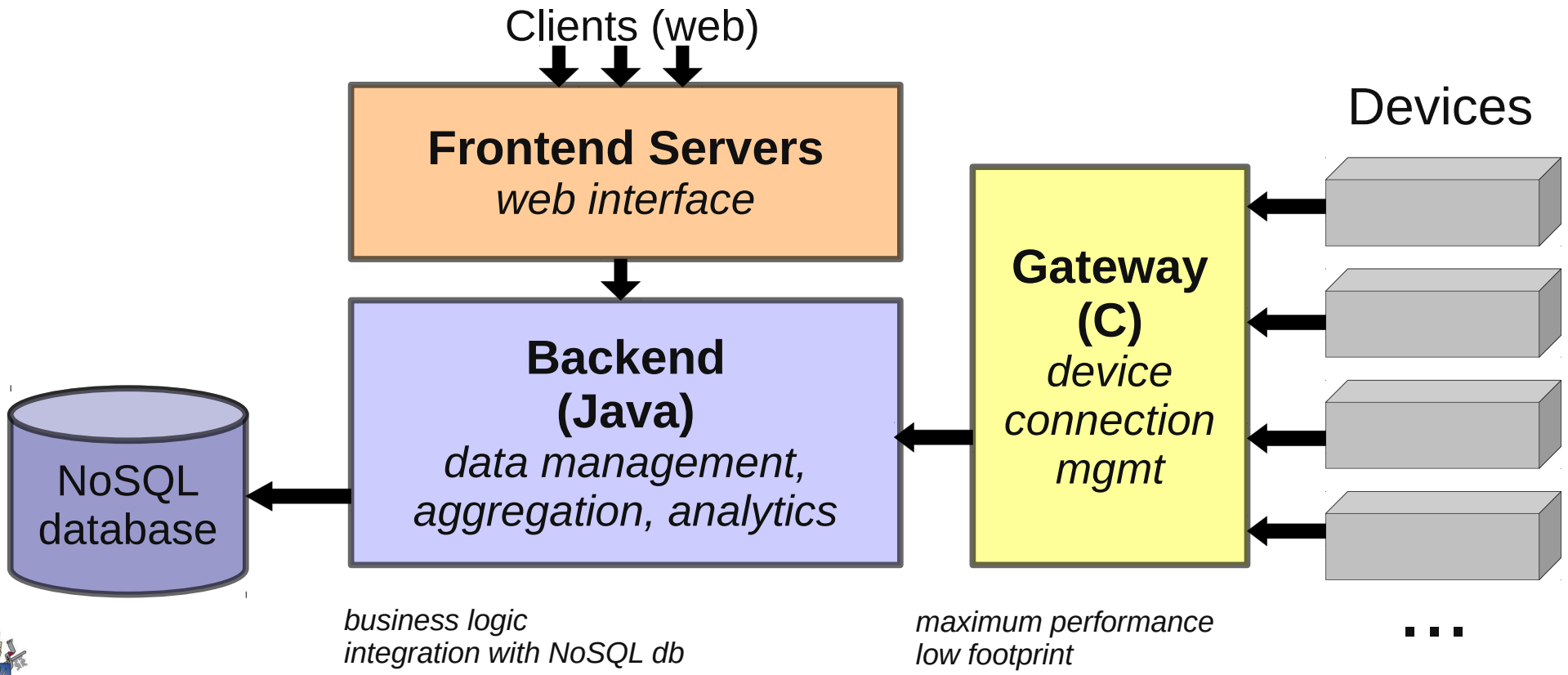
The Iplenix architecture



The Iplenix architecture

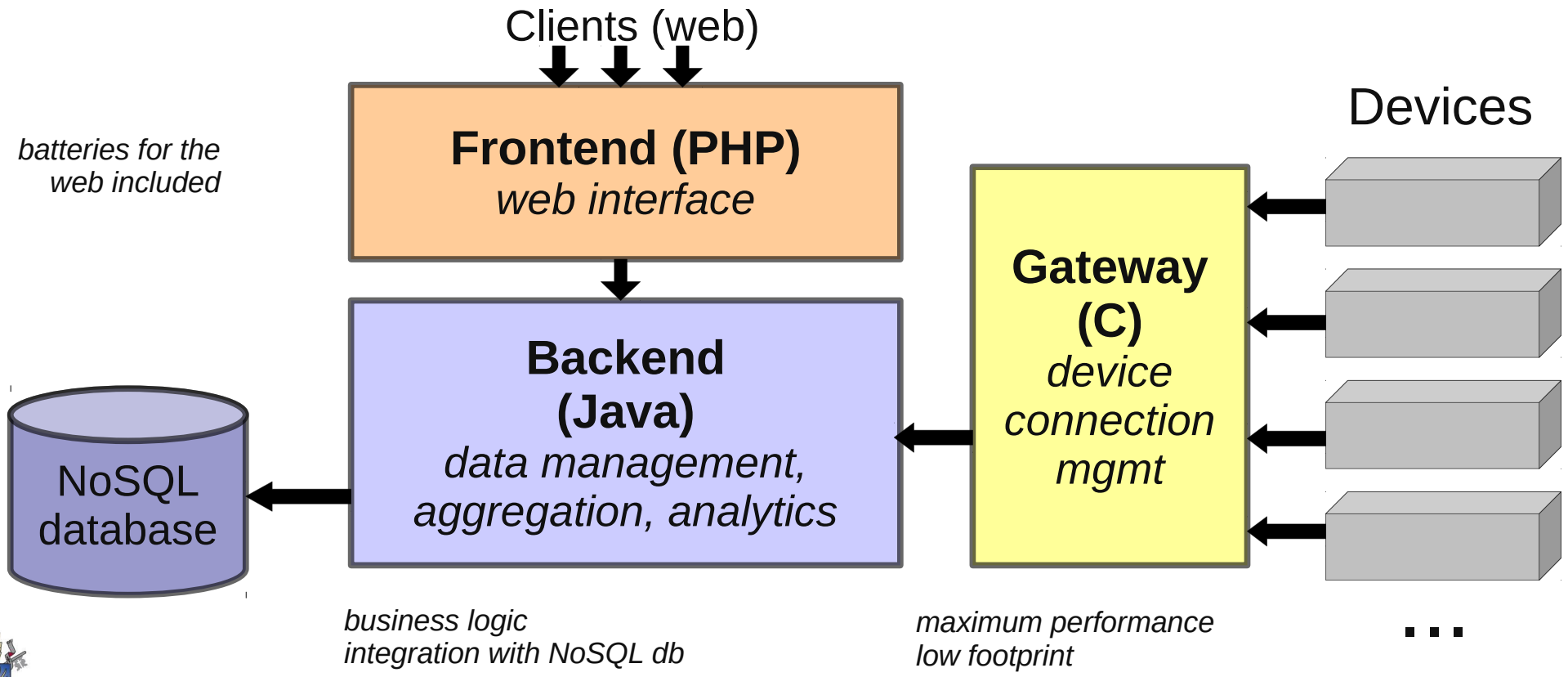


The Iplenix architecture



PUC RIO

The Iplenix architecture

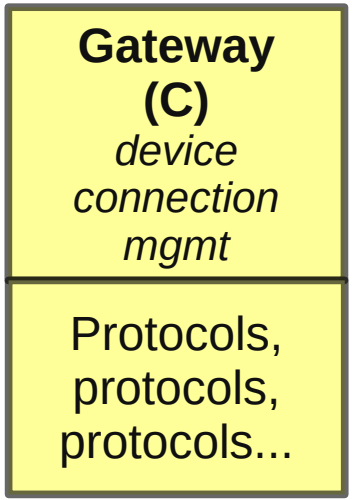


Where did Lua fit in?

- Two challenges:
  - Many incompatible devices connecting
    - Gateway needs to be quickly adaptable
  - Ever-changing energy tariffs and legislation
    - Backend needs to be quickly adaptable

The Gateway Server

- Many incompatible devices connecting
  - Generators, energy accounting devices
  - Each speaks its own protocol (or incompatible / poorly-implemented variations of protocols!)
  - Almost one protocol for each customer
  - Written in C: recompile, rebuild, redeploy, restart



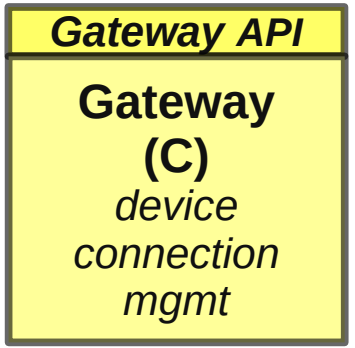
Lua in the Gateway

- Strip away the protocols

Gateway
(C)
device
connection
mgmt

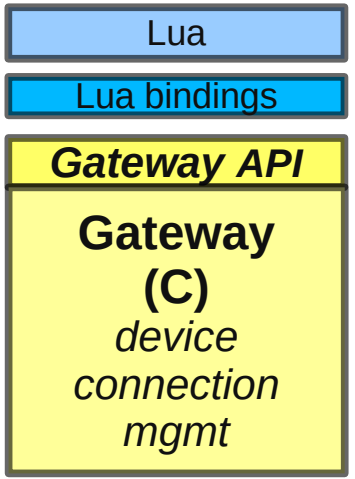
Lua in the Gateway

- Strip away the protocols
- High-level API with the main features of the gateway



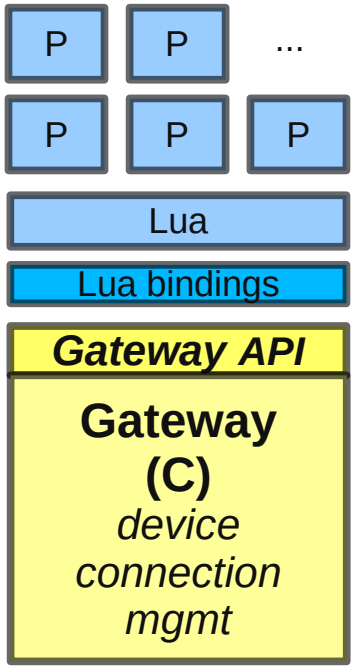
Lua in the Gateway

- Strip away the protocols
- High-level API with the main features of the gateway
- Bindings to connect the Lua VM to this API

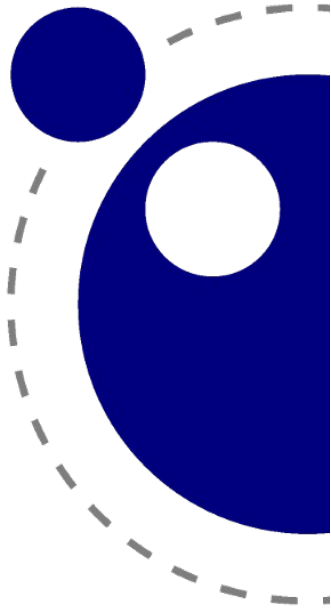


Lua in the Gateway

- Strip away the protocols
- High-level API with the main features of the gateway
- Bindings to connect the Lua VM to this API
- Protocols in Lua



What Lua bindings look like

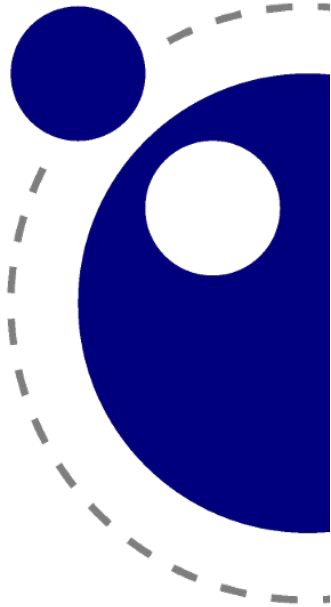


```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

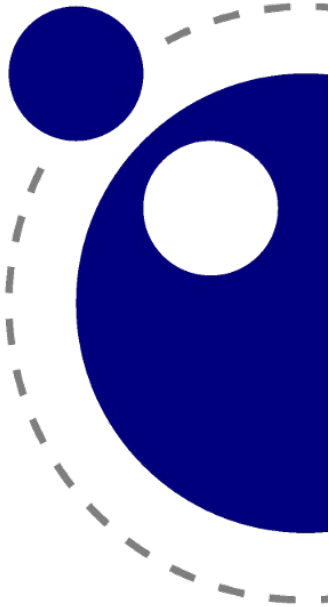
What Lua bindings look like

handle to a Lua instance (yes, there can be many)

```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```



What Lua bindings look like



read arguments from Lua into C

```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

What Lua bindings look like

we can store pointers to C context data inside our Lua context

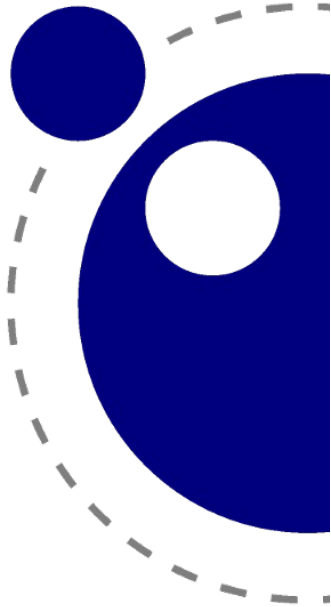
```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
    Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

What Lua bindings look like

we call our Gateway API function

```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
   sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

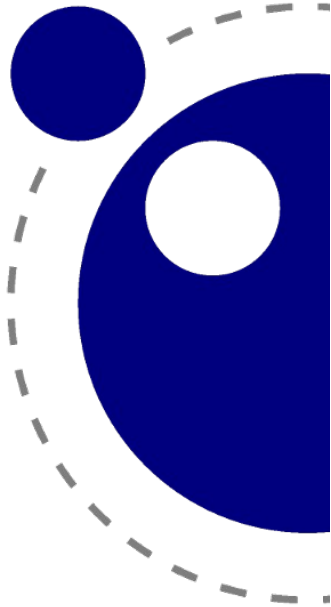
What Lua bindings look like



```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

we clean the stack and
push our return value back to Lua
(yes, there can be many)

What Lua bindings look like

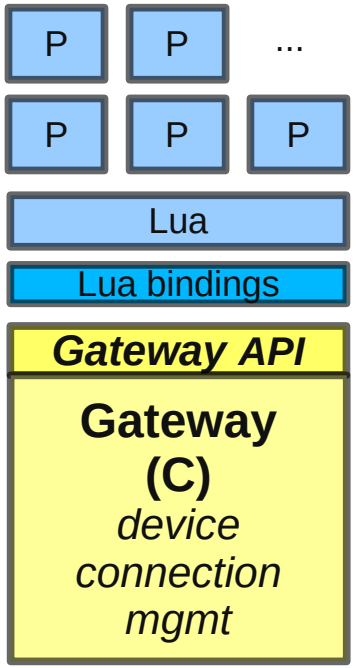


```
int send_data(lua_State* L) {
    size_t size, sent;
    const char* data = luaL_checklstring(L, 1, &size);
    lua_getfield(L, LUA_REGISTRYINDEX, "gw");
   Gateway* gw = lua_touserdata(L, -1);
    sent = Gateway_sendData(gw, data, size);
    lua_pop(L, lua_gettop(L));
    lua_pushinteger(L, sent);
    return 1;
}
```

...and yes, there are many bindings generators to automate this boring stuff! Huge libraries like Qt are bound using generators.

Advantages

- Writing protocols in Lua is simpler
- We could load/unload them on the fly without restarting
  - We did have support for loading C protocols as dynamic libraries, but with Lua code is bound per thread, not per process
- The C core stabilized to a point were we basically never touched it anymore



Second challenge: the frontend server

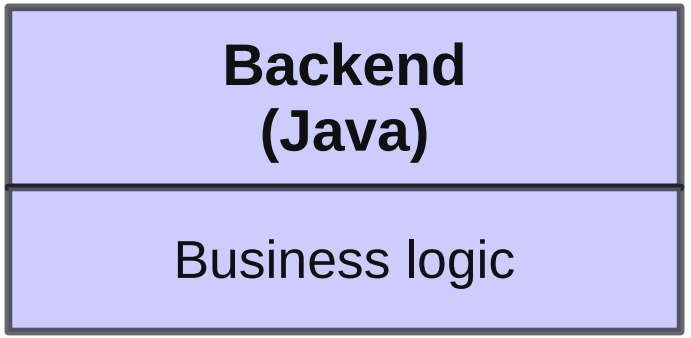
- Ever-changing business logic
  - Brazil is a large country (5th largest in both pop. and area) with a privatized energy system
  - Each of the 26 states has one or more electricity companies

AES Sul • AMAZONAS • Ampla • Bandeirante • BOA VISTA • Caiuá • CEA • CEAL • CEB • CEEE • CELESC • CELG • CELPA • CELPE • CELTINS • CEMAR • CEMAT • CEMIG • CEPISA • CERON • CERR • CFLM • CFLO • Chesp • CJE • CLFSC • CNEE • COCEL • Coelba • COELCE • COOPERALIANÇA • COPEL • COSERN • CPEE • CPFL Paulista • CPFL Piratininga • CSPE • DEMEI • DMED • EBO • EDEVP • EEB • EFLUL • ELEKTRO • ELETROACRE • ELETROCAR • AES Eletropaulo • ELFJC • ELFSM • EMG • ENERSUL • ENF • EPB • Escelsa • ESE • FORCEL • HIDROPAN • IGUAÇU • Light • MUXFELDT • RGE • SULGIPE • UHENPAL

  - Billing plans and rates change, laws change
  - Also, data from devices also had to be processed in different ways

The frontend server

- Responsible for several tasks
  - Receiving data from gateway servers
  - Responding requests from frontend servers
  - Taking data in and out of the NoSQL database
  - The big part that kept changing, though, was the business logic



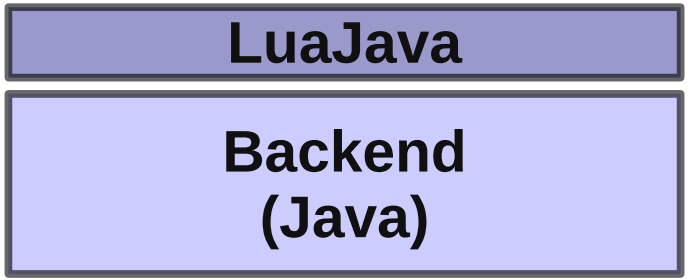
Lua in the backend

- Strip away the business logic

Backend (Java)

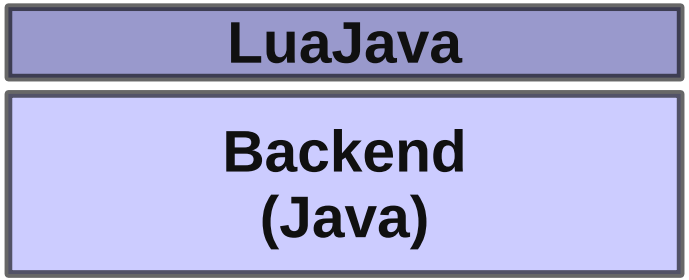
Lua in the backend

- Strip away the business logic
- Plug in the LuaJava bridge



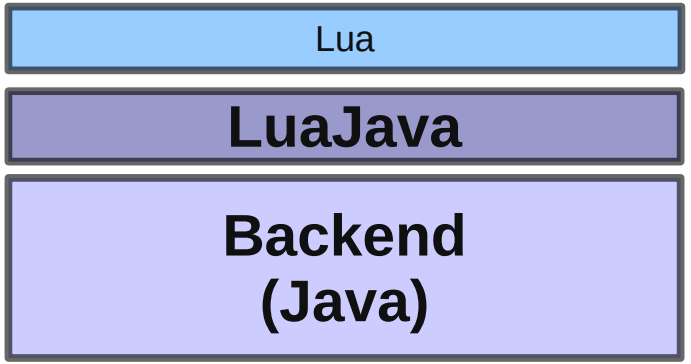
Lua in the backend

- Strip away the business logic
- Plug in the LuaJava bridge
- LuaJava uses the Java Reflection API which discovers classes automatically: no bindings needed!



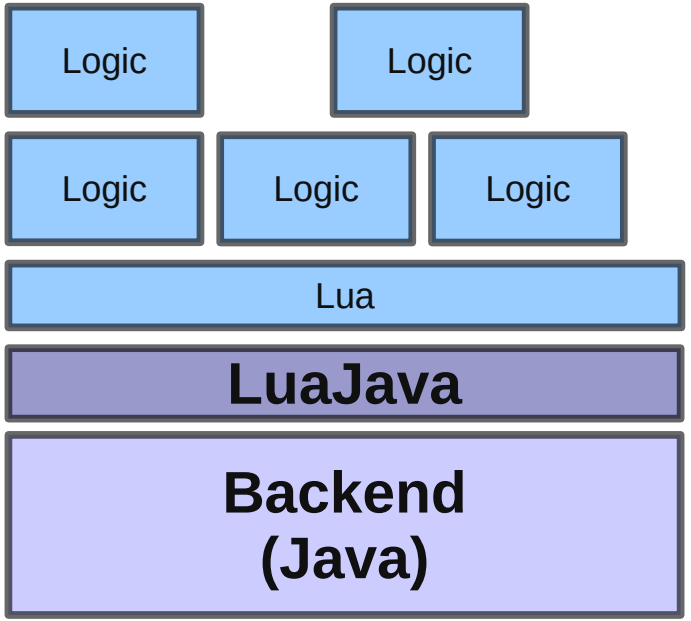
Lua in the backend

- Strip away the business logic
- Plug in the LuaJava bridge
- LuaJava uses the Java Reflection API which discovers classes automatically: no bindings needed!



Lua in the backend

- Strip away the business logic
- Plug in the LuaJava bridge
- LuaJava uses the Java Reflection API which discovers classes automatically: no bindings needed!
- Write business logic in Lua



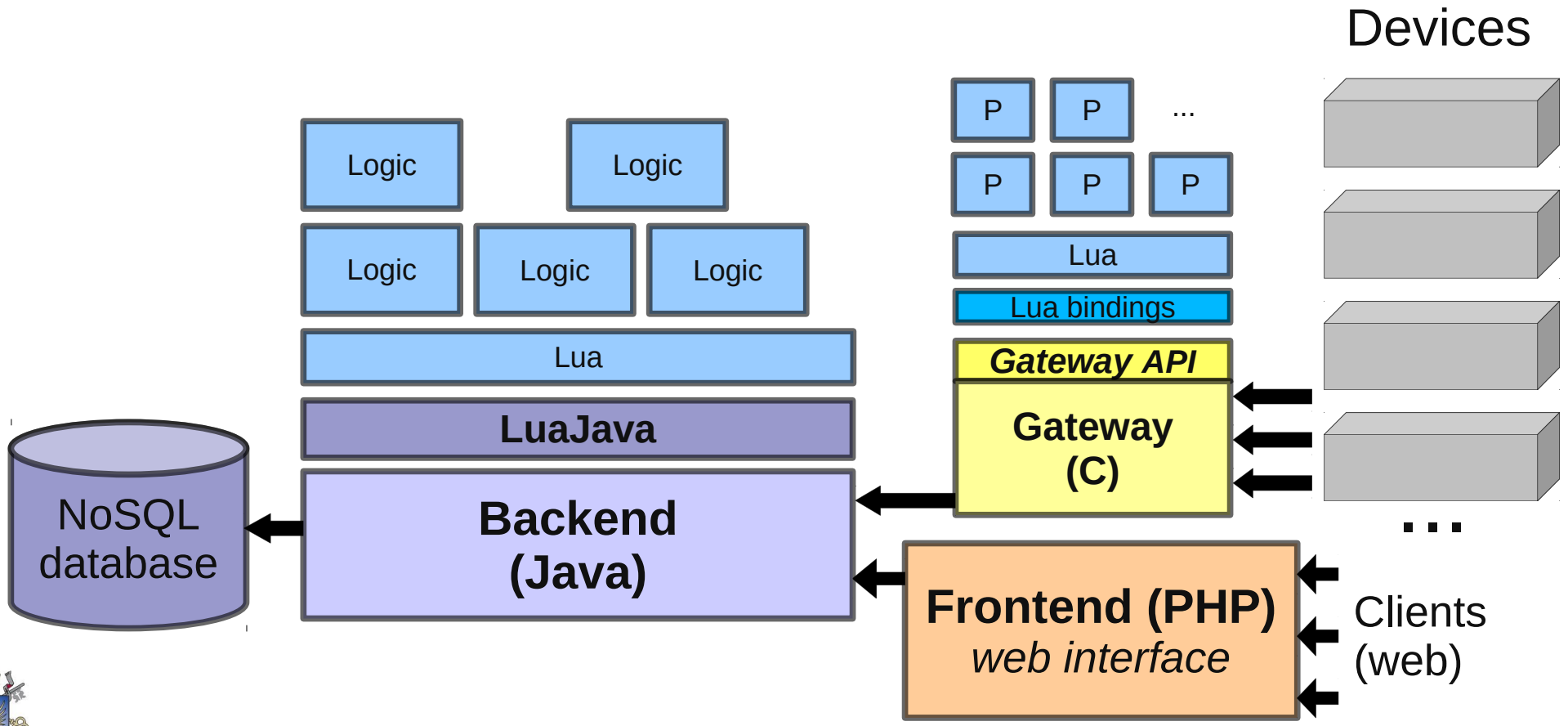
Advantages

- Simpler to write business code logic
- On-the-fly updates
- We wrote some specialized, high-level Java classes with simplified interfaces to be used from Lua
- We also wrote some custom Lua code to hide some Java-isms:

```
for obj in each(vec) ... end
```

A 5-line function which implements a Lua iterator that accepts any Java objects that implements `Iterable`.

The Iplenix architecture, improved



Results

- A very flexible solution
- We tried to use, for each job, the best tool for the job
- The message is: don't be afraid to integrate languages
  - Give programmers the most adequate tools
    and they will be more productive
- When I joined Iplenix, I was the only one who knew Lua
- When I left, at least five others in the team were familiar with it
  - Also using it for other tasks

Some high-profile uses of Lua

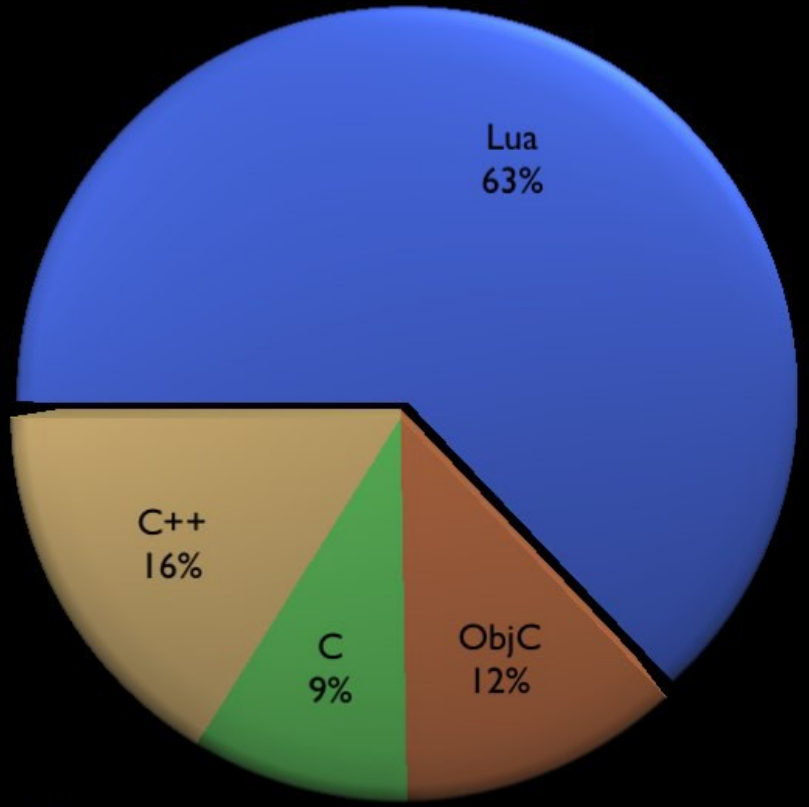
- That was just one real-world example
- Lua is used in many industrial-strength applications,
  both commercial and open source

- Adobe Photoshop Lightroom
- Apache web server
- VLC media player
- Corona SDK for iOS/Android
- Cisco Adaptive Security Appliance
- Many others we'll never know about

How Much Do We Use Lua?

http://troygaul.com/LrExposedC4.html

- A lot!
  - 63% of the main Lightroom-team authored code is Lua



Lua in games

- Typical split between "heavy lifting/backend" (engine) and "~~ business~~ *game* logic"
- World of Warcraft, a massively multiplayer online RPG
  - UI is customizable in Lua
  - tweaked by power users
- Angry Birds, casual game for mobile
  - Levels written in Lua
  - data files stored in Lua (save games, high scores)

Allods Online - American Girl - Angry Birds - Aquaria - Baldur's Gate - The Battle for Wesnoth - Bet On Soldier: Blood Sport - Blitzkrieg - Brave: The Search for Spirit Dancer - Brütal Legend - Bubble Ball - Buzz! - BZFlag - Civilization V - Company of Heroes - Cortex Command - Crackdown - Crowns of Power - Crysis - DarkSpace - Dead Hungry Diner - Demigod - Digital Combat Simulator - Diner Dash - Driver: San Francisco Dungeon Crawl Stone Soup - Dungeons - Empire: Total War - Enigma - Escape from Monkey Island - Etherlords - Eufloria - Evil Islands: Curse of the Lost Soul - EXperience112 - Fable II - The Fairly OddParents Shadow Showdown - Far Cry - FlatOut - FlatOut 2 - Foldit - Fortress Forever - Freeciv - Freeciv Greatturn - Freelancer - Garry's Mod - Grim Fandango - The Guild 2 - Tom Clancy's H.A.W.X - Headhunter Redemption Hearts of Iron III - Hedgewars - Heroes of Might and Magic V - Homeworld 2 - Hyperspace Delivery Boy! - Impossible Creatures - The Incredibles: When Danger Calls - King's Bounty: The Legend - L.A. Noire - Legend of Grimrock - Lego Universe - Linley's Dungeon Crawl - Lock On: Modern Air Combat - Mafia II - Magic: The Gathering – Duels of the Planeswalkers - MDK2 - Mercenaries: Playground of Destruction - Metaplace - Monopoly Tycoon - Multi Theft Auto - MUSHclient - Napoleon: Total War - Natural Selection 2 - Operation Flashpoint: Dragon Rising - Orbiter - Painkiller - PlayStation Home - Project Zomboid - Psychonauts - Puzzle Quest: Challenge of the Warlords - Rail Simulator - RailWorks - Rappelz - Regnum Online - Requiem: Memento Mori - Richard Burns Rally - Rift - RigidChips - Roblox - Rolando 2: Quest for th Golden Orchid - Room for PlayStation Portable - ROSE Online - Runes of Magic - Ryzom - S.T.A.L.K.E.R.: Shadow of Chernobyl - Saints Row 2 - Serious Sam 3: BFE - Shank - Shank 2 - Silent Storm - SimCity 4 - The Sims 2: Nightlife - Singles: Flirt Up Your Life - Skyland SSR - Sonic Unleashed - SpellForce: The Order Dawn - SplitApple - Spring - Star Wars: Battlefront - Star Wars: Battlefront II - Star Wars: Empire at War - Sta Wars: Empire at War: Forces of Corruption - Star Wolves - StepMania - Stolen - Stratagus - Strike Suit Zero Supreme Commander - Supreme Commander: Forged Alliance - T-80 Darts - Tales of Pirates - Tap Tap Revenge - There - Toribash - Trouble in Terrorist Town - ÜberSoldier - UFO: Afterlight - UltraStar - Universe a War: Earth Assault - Vegas Tycoon - Vendetta Online - Warhammer 40,000: Dawn of War - Warhammer 40,000: Dawn of War II - Widelands - The Witcher - World of Warcraft - X-Moto - You Are Empty

..plus many others

One recent adoption of Lua...

One recent adoption of Lua...

Lua in Wikipedia

- Lua has been recently chosen as the scripting language for Wikipedia
- It will gradually replace the existing custom template language
- Templates create infoboxes, summaries, [*citation needed*] links, etc.
- Some Wikipedia pages take up to 30 seconds to render due to the old template language! (and you thought your internet was slow)

PUC RIO

Lua in Wikipedia

- These scripts are written *by the users*, right from their browsers, and run in the server
- Major demonstration of Lua's safety running untrusted, unverified code
- This is possible because Lua can be 100% sandboxed
  - Also features resource limits for memory and running time

PUC RIO

Lua in Wikipedia

- WikiMedia Foundation developers wrote Lua libraries to extend Lua for their needs
  - Unicode handling libraries
  - PHP integration (MediaWiki is written in PHP)
- Benchmarks:
  - PHP-to-Lua: 2µs
  - Lua-to-PHP: 0.5µs
  - As fast as PHP-to-PHP

PUC RIO

Why not other languages?

- Why not JavaScript with V8?
  - Not safe for embedding: no mem alloc hook
- Why not JavaScript with Rhino?
  - Slow startup, no CPU limit control
- Why not PHP itself?
  - Very big language, impossible to sandbox
- The Lua source code was entirely audited by the WikiMedia Security Team

PUC RIO

In conclusion...

- Lua is a mature and proven language
  - This year it celebrates its 20th anniversary!
- From Angry Birds to Wikipedia, Lua is everywhere
  - You might not knowing, but you may have been using it already!
- There are all kinds of resources available to learn Lua
  - Books, IDEs, Eclipse plugin, very active and friendly list, Reference Manual (in Russian too!)

Thank you!

Questions?

- Contact:

Lua: http://lua.org/
List: http://www.lua.org/lua-l.html

LuaRocks: http://luarocks.org/
List: http://www.luarocks.org/en/Mailing_list

me (Hisham): http://hisham.hm/
Email: h@hisham.hm

PUC RIO

About these slides...

- Feel free to share this presentation
  and to use parts of it in your own material
- Licensed under the Creative Commons CC BY 2.0:
  - https://creativecommons.org/licenses/by/2.0/

PUC RIO