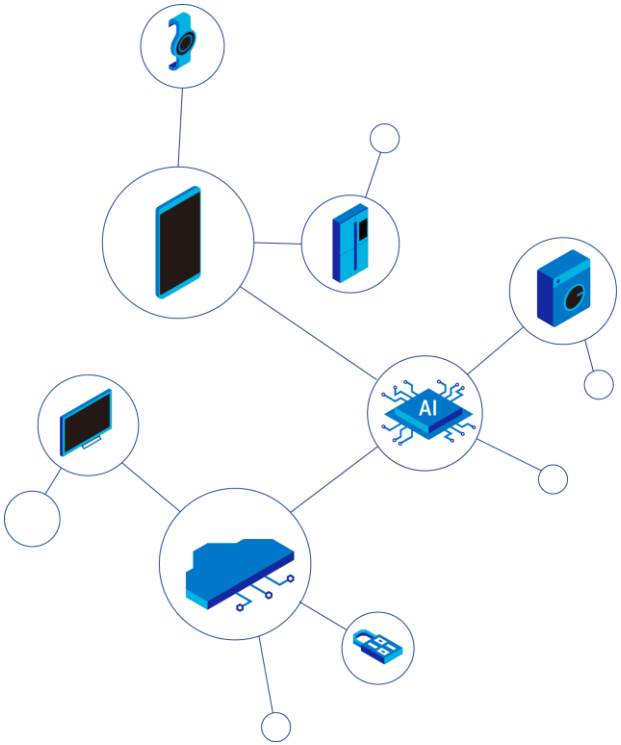


CASTANETS:

Resource Efficient Distributed Web Browser by Offloading Processes to Remote Edge Devices

https://samsung.github.io/Castanets/
https://github.com/Samsung/Castanets

Samsung Research HQ:
- Eun N.K
- Seikwon Kim
- In-soon Kim
- YongGeol Jung

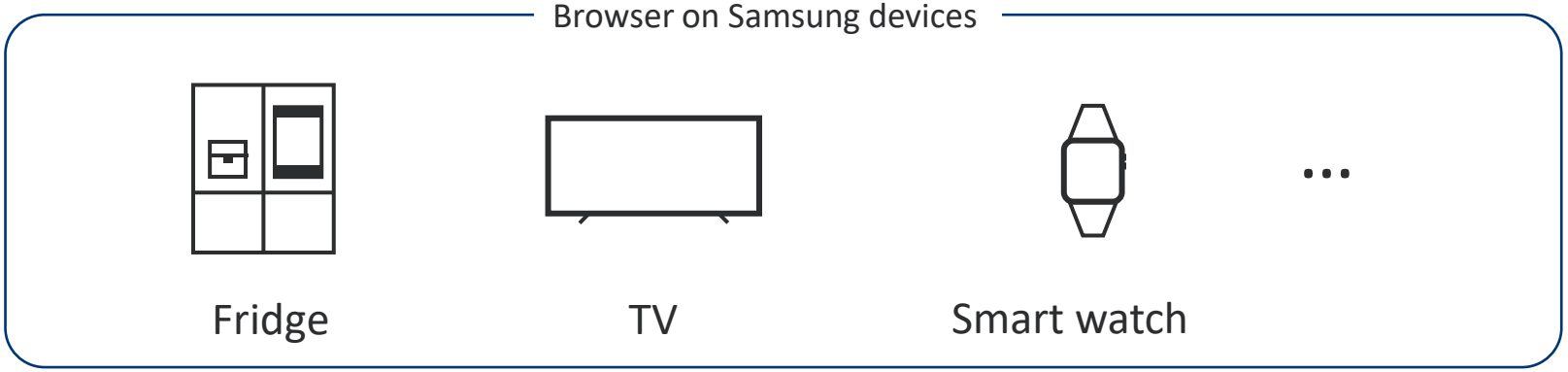
Samsung Research Bangalore:
- Nagarajan Narayanan
- Venu Madhav Musham
- Uzair Jaleel
- Chandan Padhi
- Venu Gopal SM
- Suyambulingam Rathinasamy Muthupandi

SAMSUNG Research

Web Technology Trends

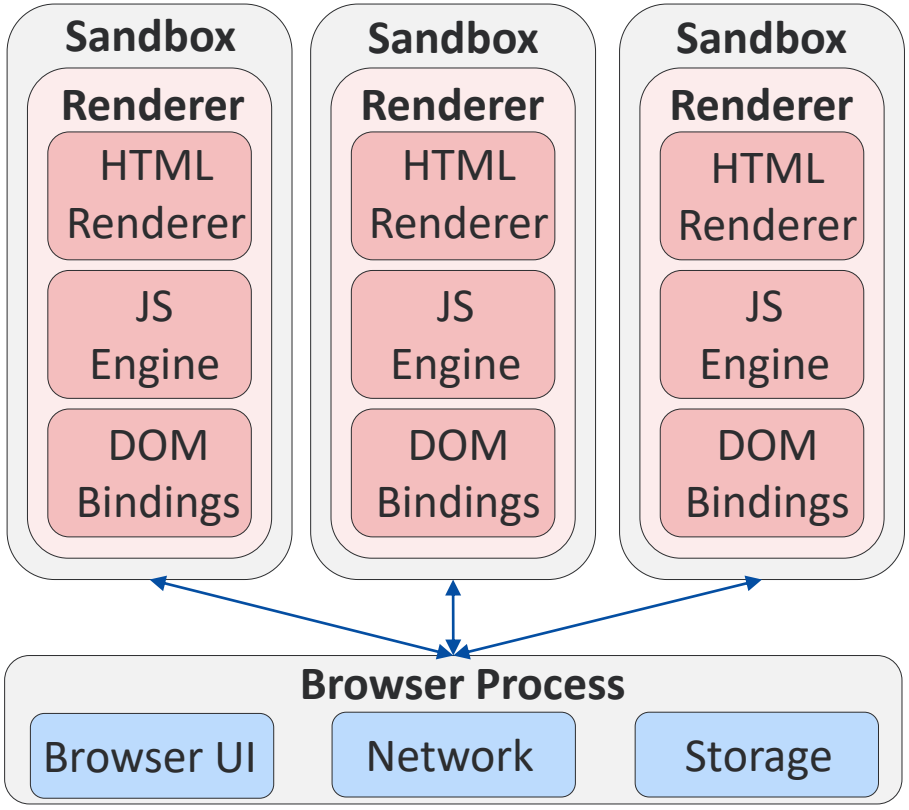
- Web contents getting more complicated
  - Multi-layers
  - High quality effects and animations
- Increasing browser resource consumption
  - High CPU utilization for tree generation
  - High memory consumption
  - Increasing browser binary size
- Browser everywhere on any device
  - PC → smart devices → low-end devices

Conflict!



Modern Chromium Browser

- Multiprocess architecture
  - One browser process: More graphics bound
  - Multiple renderer processes: More CPU and memory bound



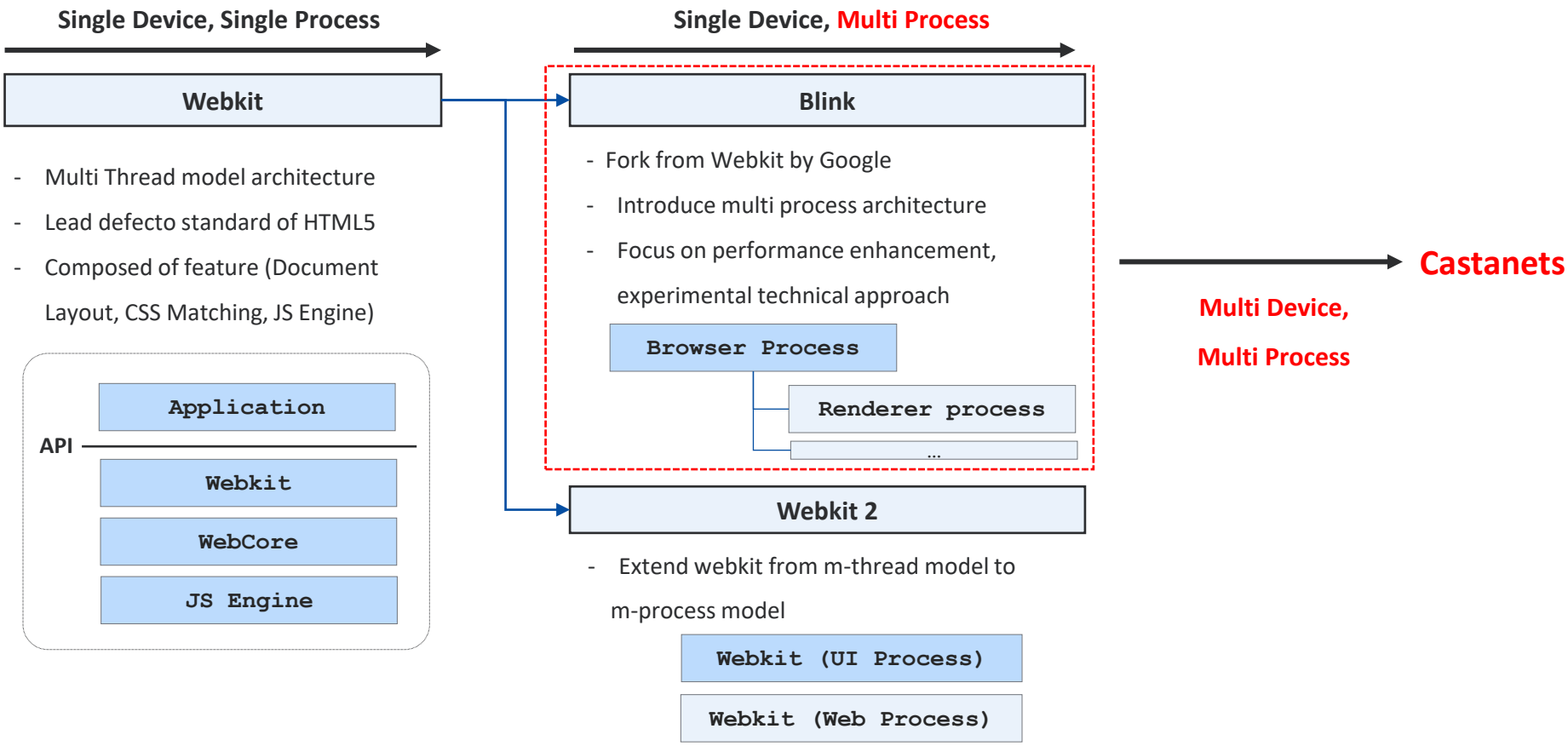
In This Talk,

Supporting a browser
on low-end devices
in the edge computing era

Castanets (Cast.a.nets)

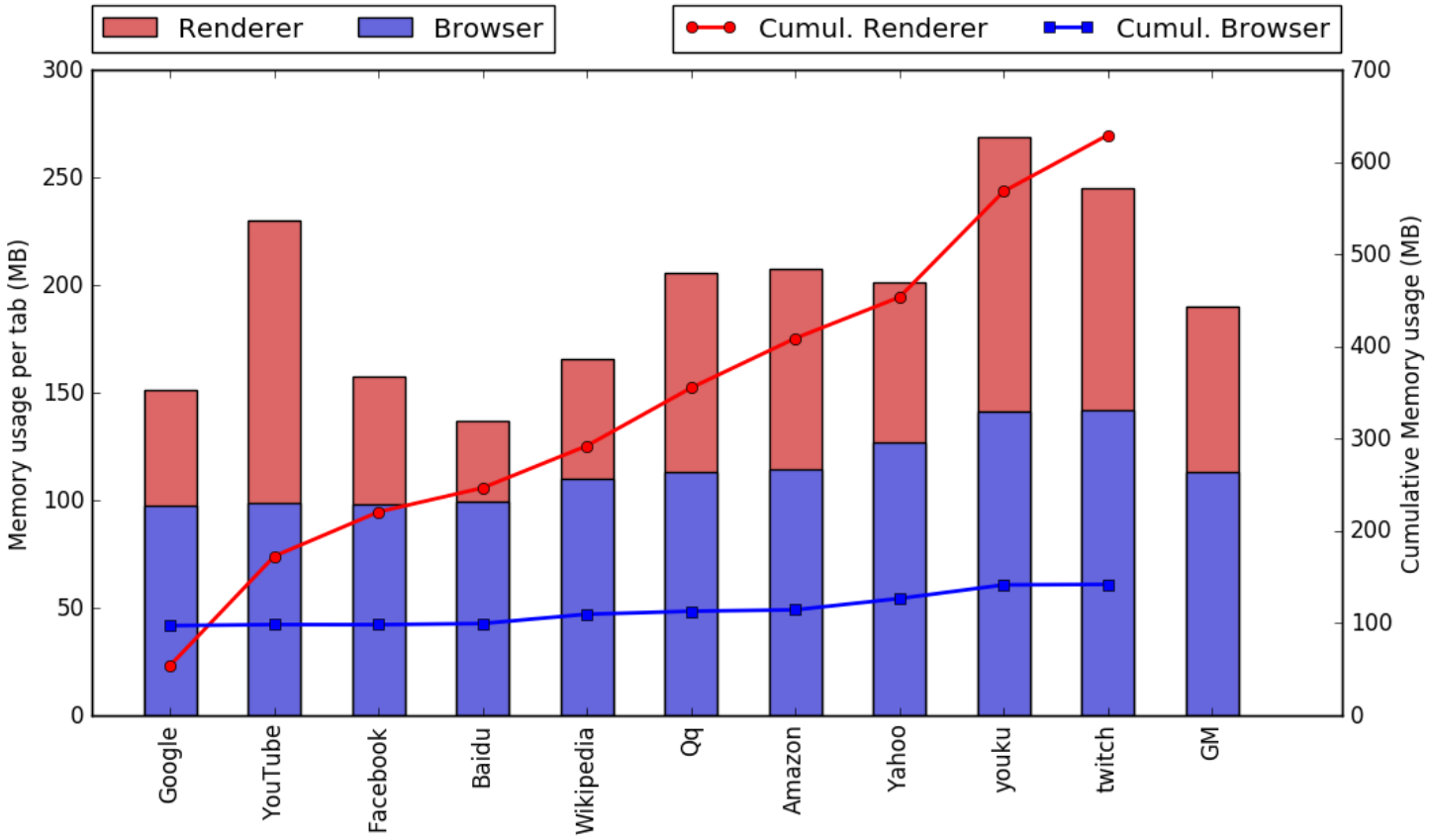
Past, Present, Future Web Engine

Multi device extended Blink



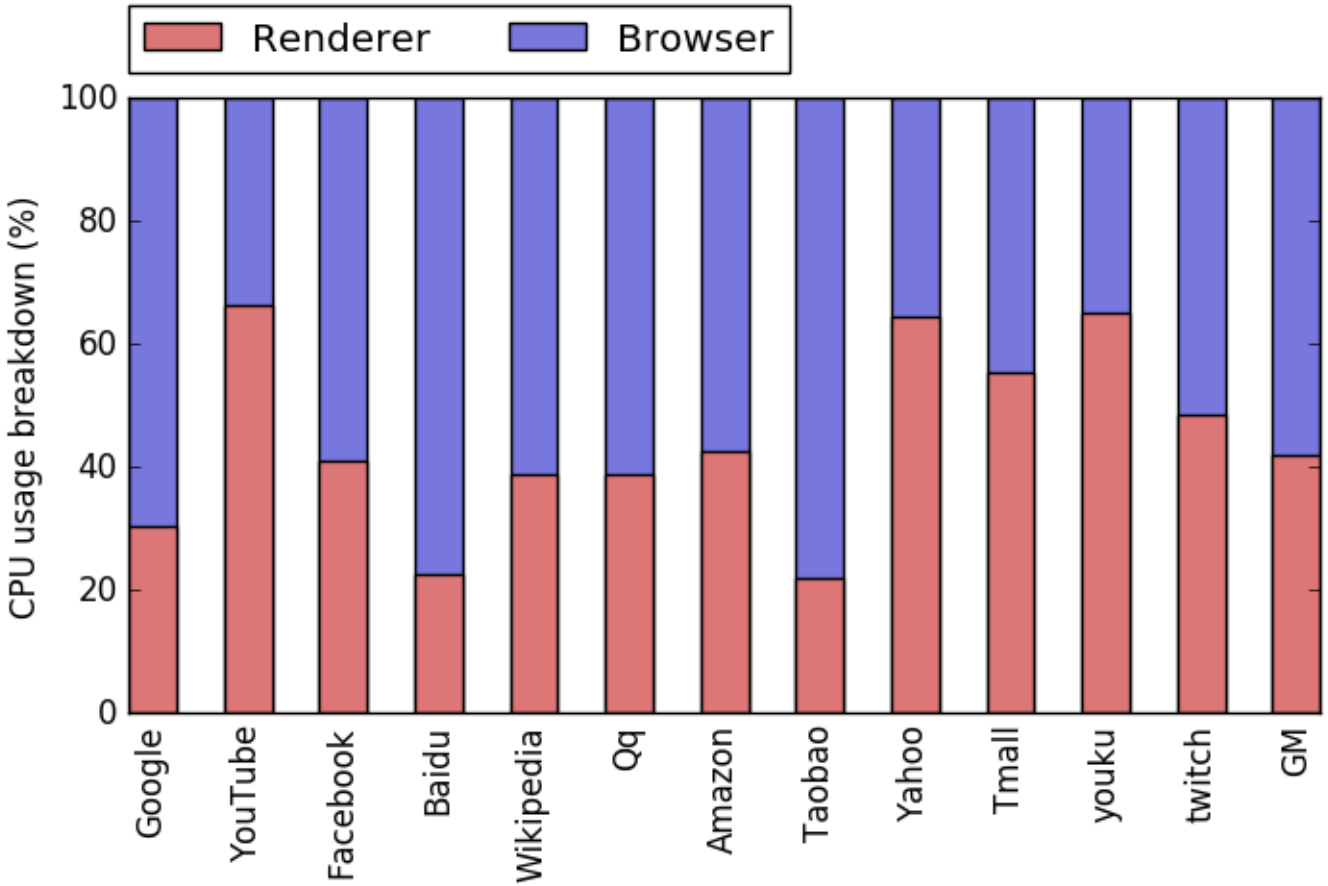
Potential Memory Usage Reduction

- Average memory usage
  - Renderer process: ~100MB
  - Browser process: ~150MB
- Renderer process memory cumulative as tabs open: 1145% with 10 tabs



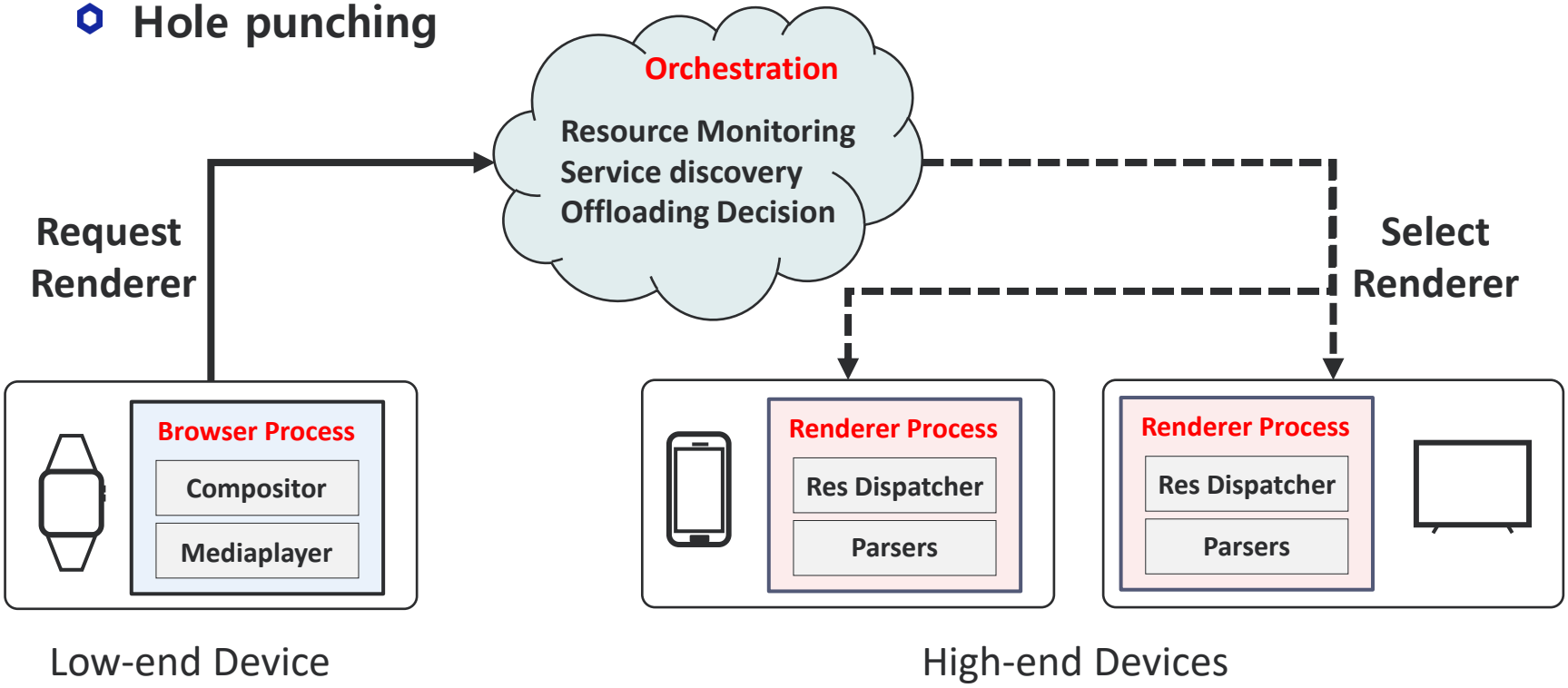
Potential CPU Usage Reduction

- Average CPU usage breakdown
  - Renderer: 42%
  - Browser: 58%



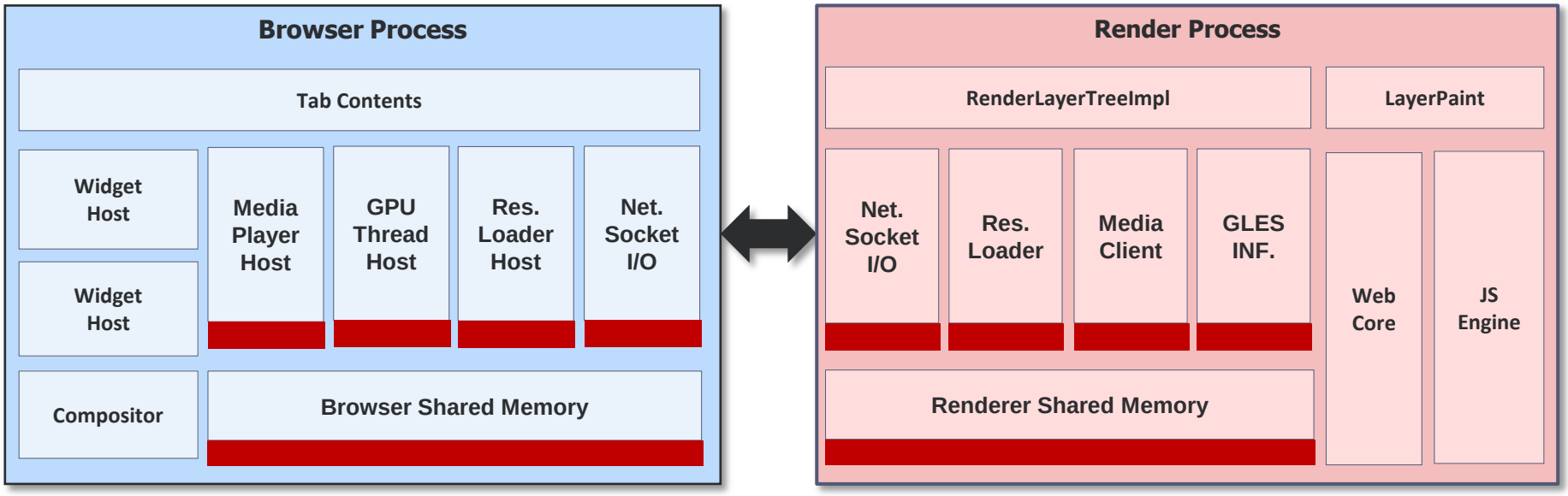
Castanets In a Nutshell

- Save HW resource of target device by distributing process
- Low-end local node: browser process
- High-end remote node: renderer process
- Orchestration server
  - Service discovery
  - Load balancing
  - Hole punching



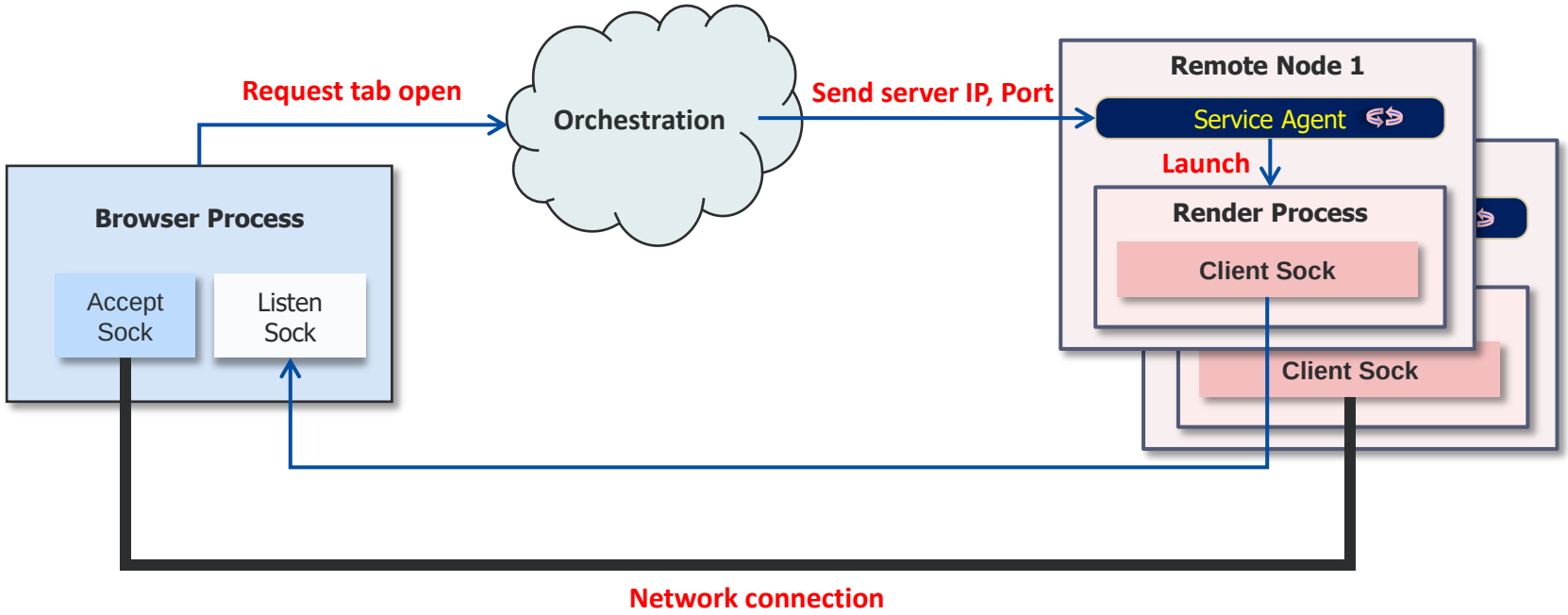
Major Changes in Castanets

- Module extensions
  - Resource loader
  - Multimedia pipeline
  - GPU Acceleration
  - And so on...



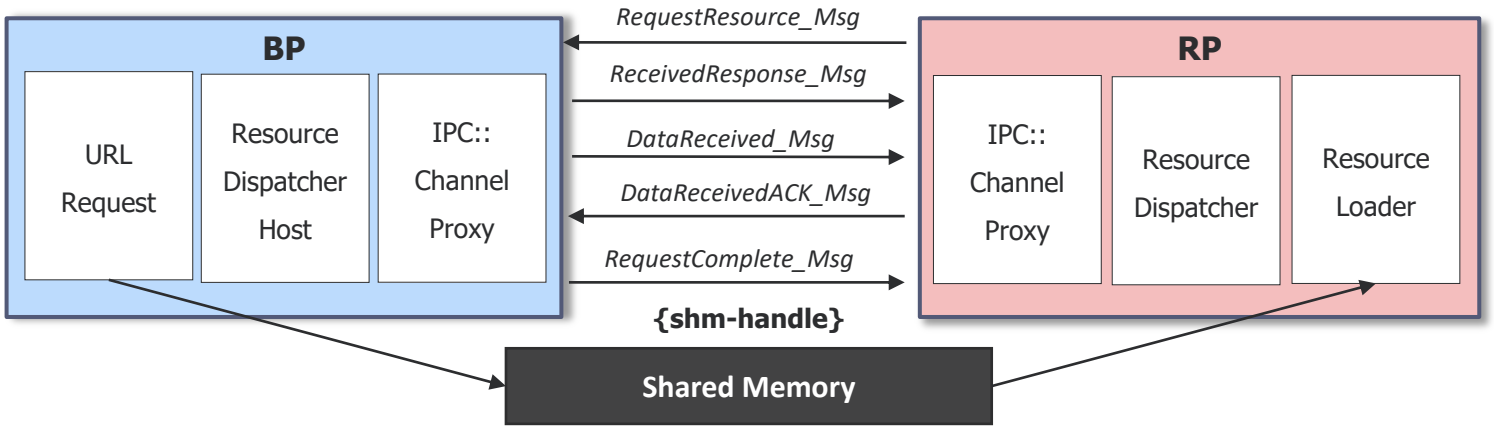
Castanets Launching Procedure

- Server: Browser process
- Client: Renderer process
- Daemon: Waits for connection, launches renderer process



Chromium Resource Loader

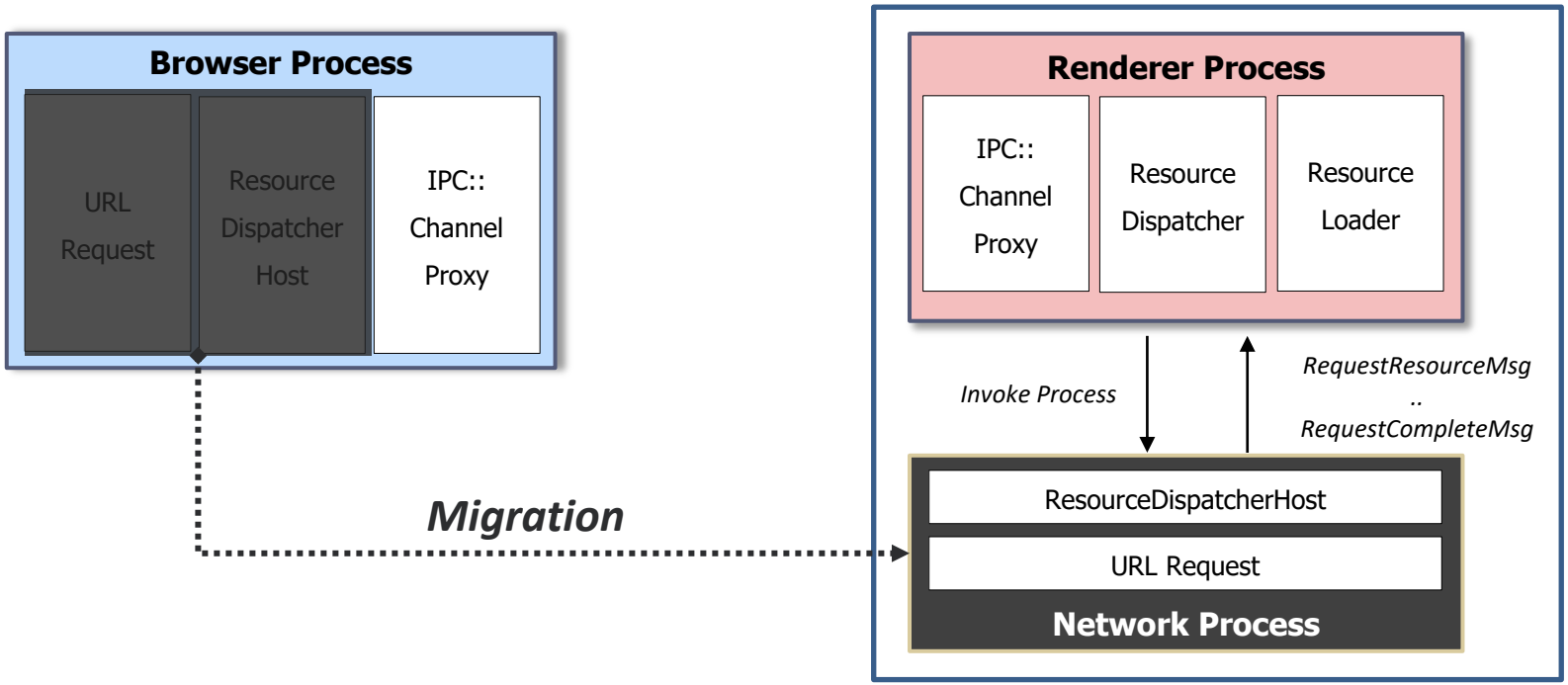
- Resource loading process in Chromium
  1. RP requests resources to BP
  2. BP fetches resources via network
  3. BP writes to shared memory
  4. BP sends shared memory handler to RP



* RP: Renderer Process
* BP: Browser Process

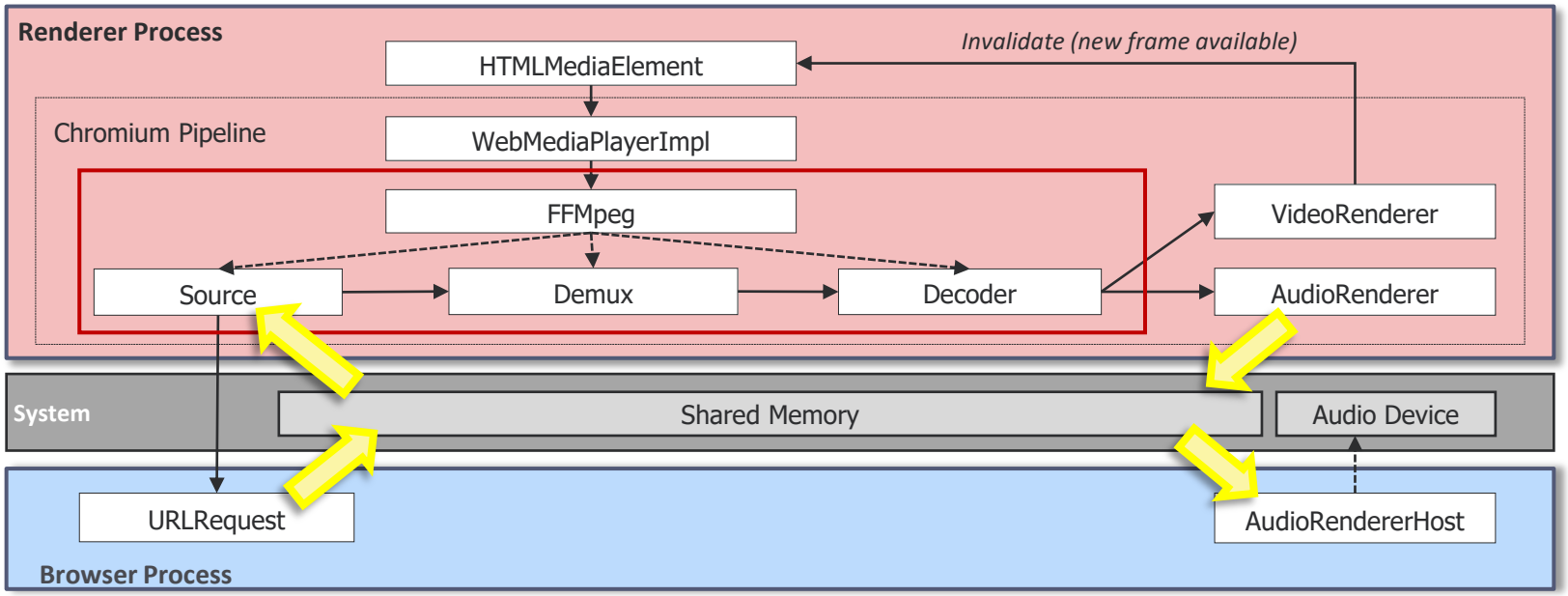
Castanets Resource Loader

- Remote device resource loading
  - Performance improvement
  - Reduce extra network traffic



Chromium Multimedia Support

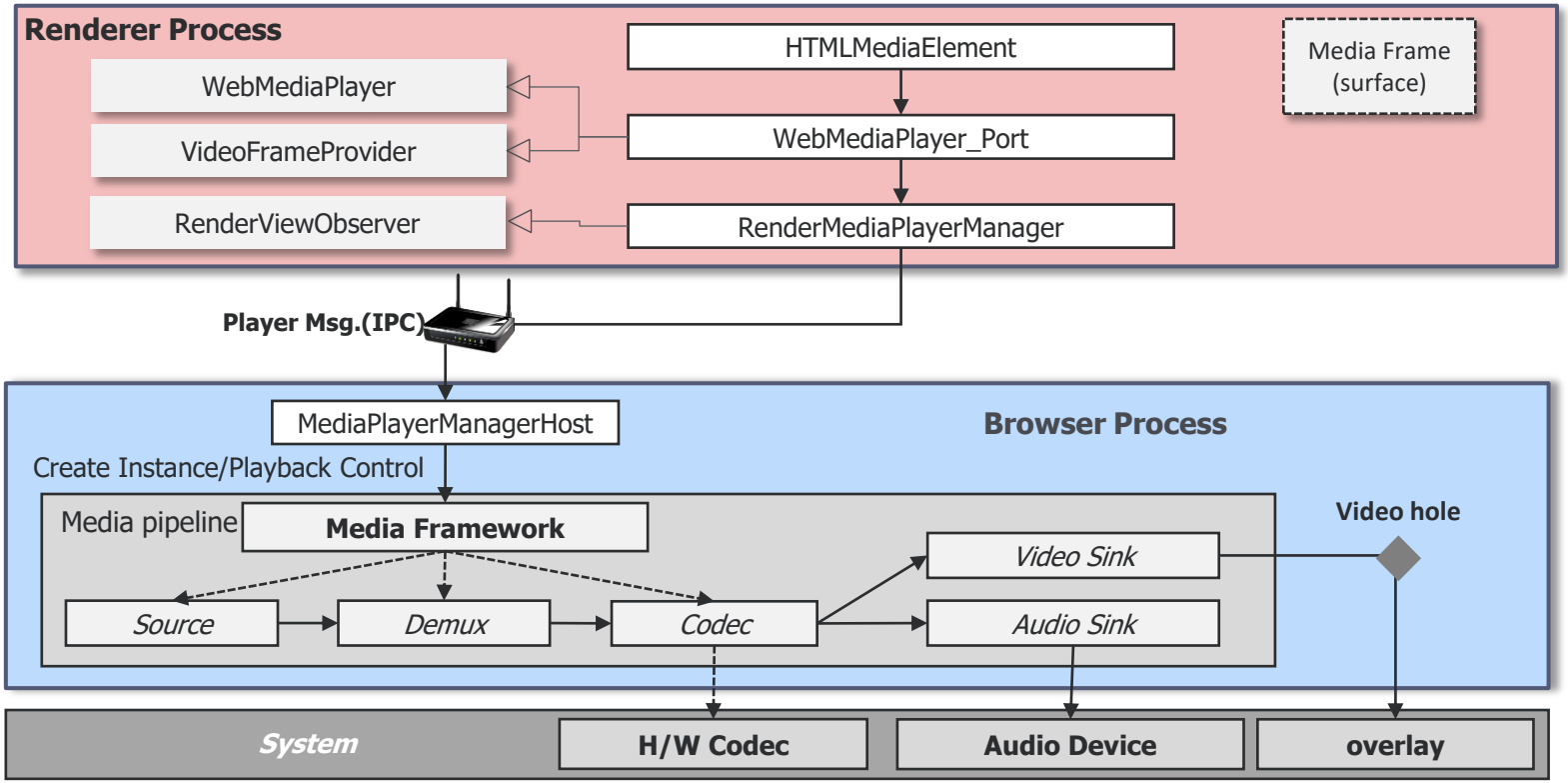
- According to global Alexa Top 50:
  - Youtube – 2nd
  - Twitch – 26th
- Original Chromium generates media pipeline in RP



Media pipeline in Chromium

Castanets Multimedia Support

- Performance issues of distributing RP and BP for multimedia support
  - BP media file transfers to RP via network
  - Decoded media stream transfers to BP via network
- Castanets renders media in BP
  - Reduces performance penalty



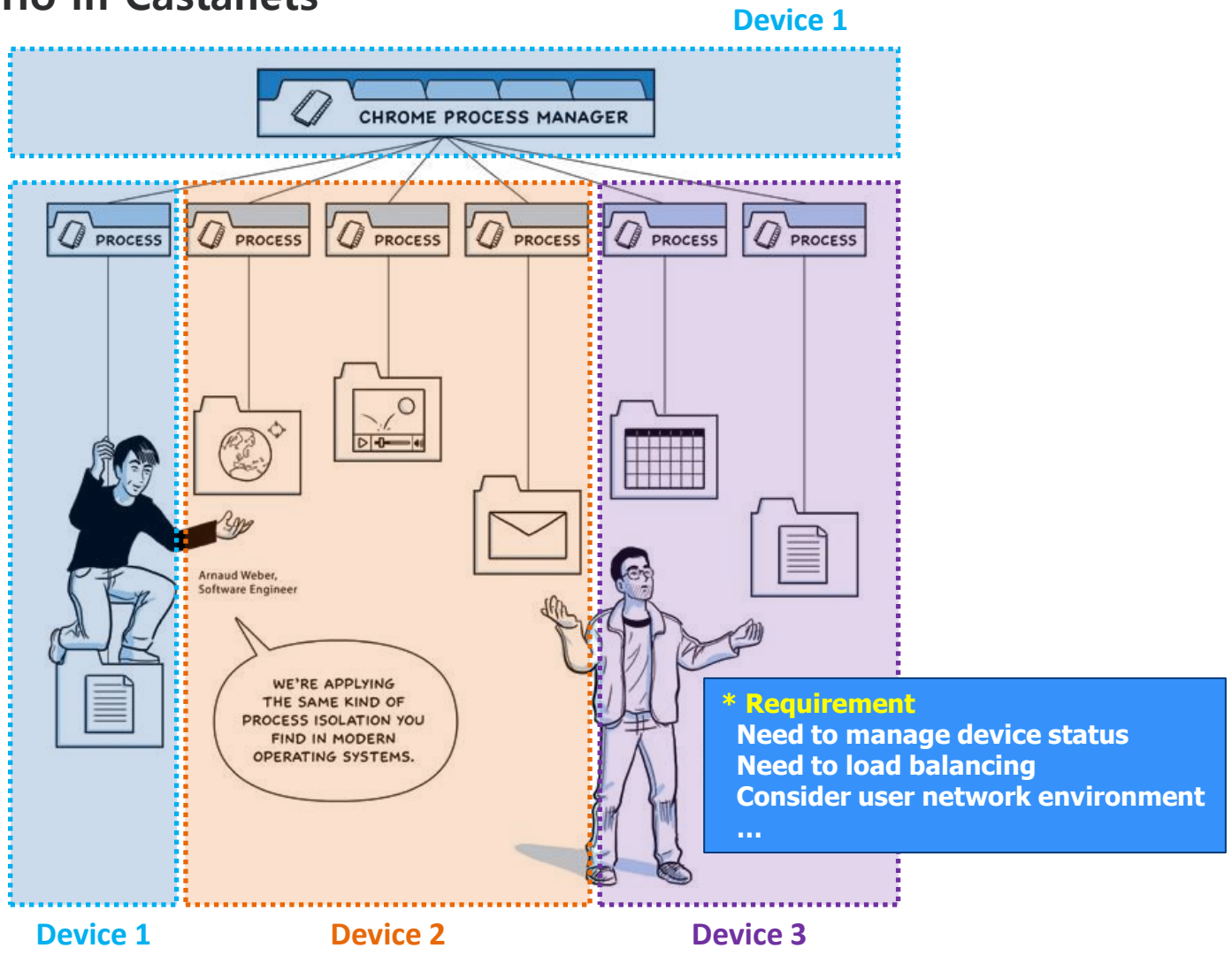
Limitations and Issues

- Communication
  - Between heterogeneous web engines
  - Between different versions
    - → IPC message with different parameter

- Network problem
  - Disconnection while JS execution, sending tiles, and so on
  - Renderer device turns off
  - All other kinds of disconnections

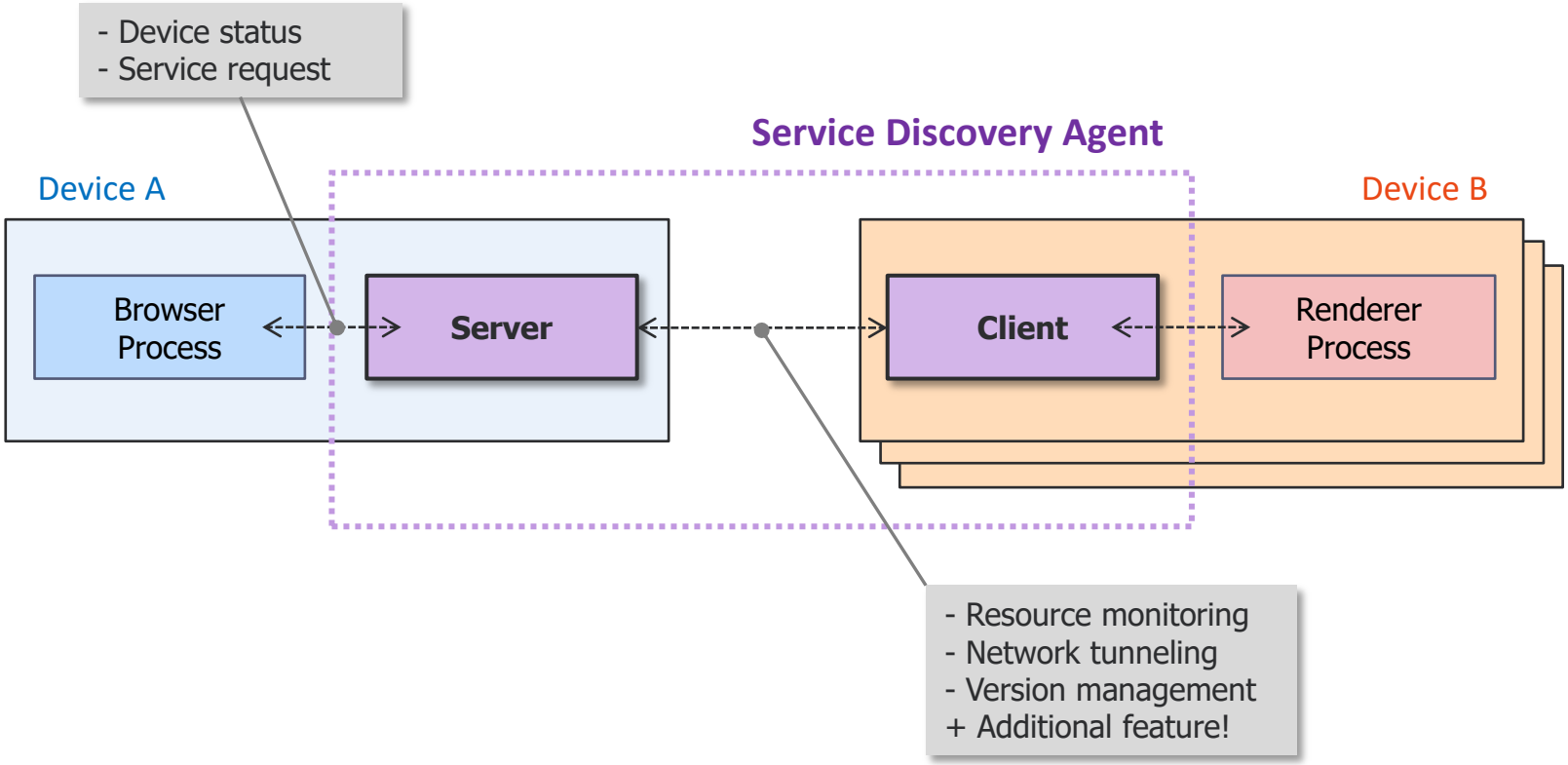
What is Service Discovery Agent?

- User scenario in Castanets



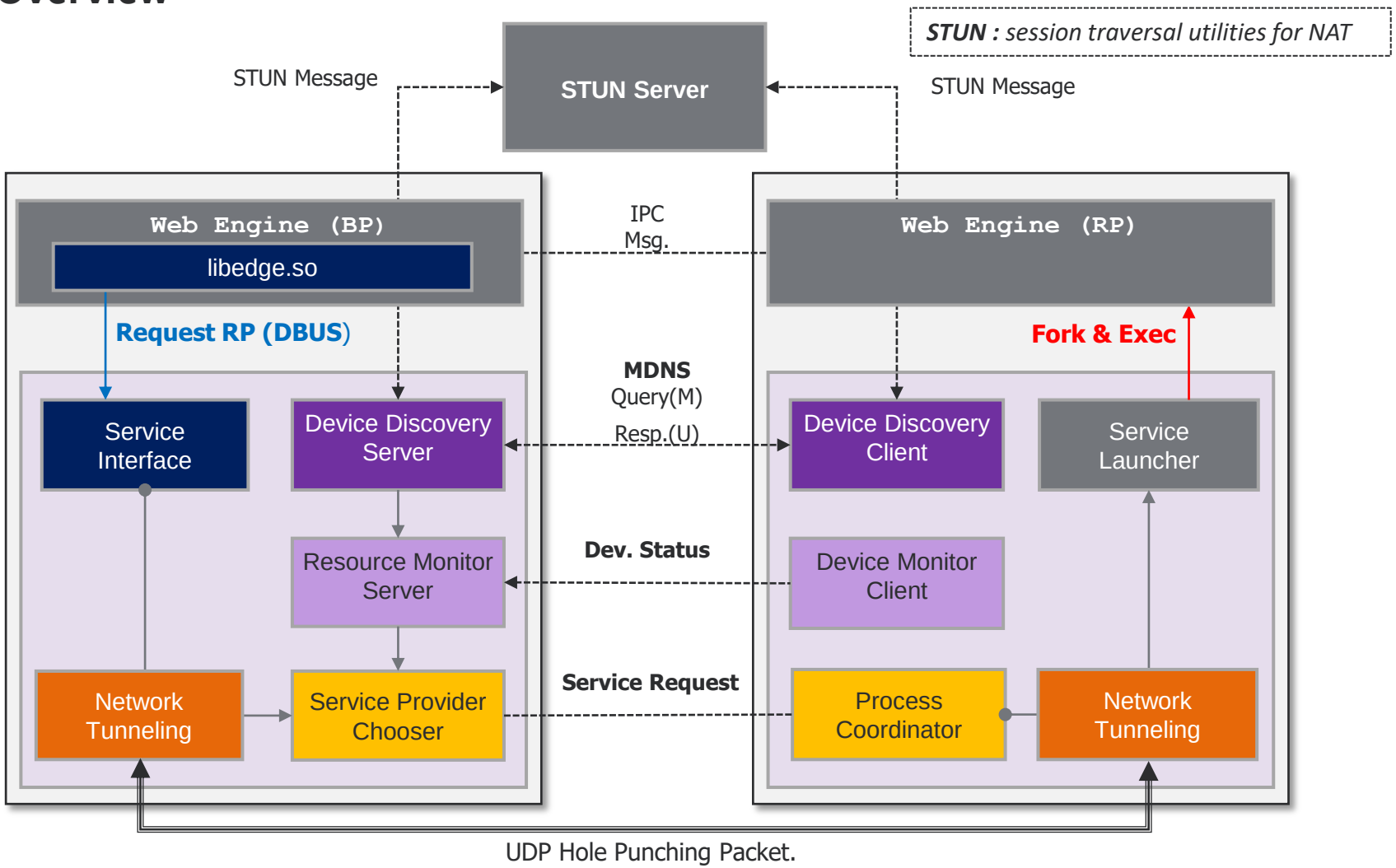
What is Service Discovery Agent?

Service Discovery Agent



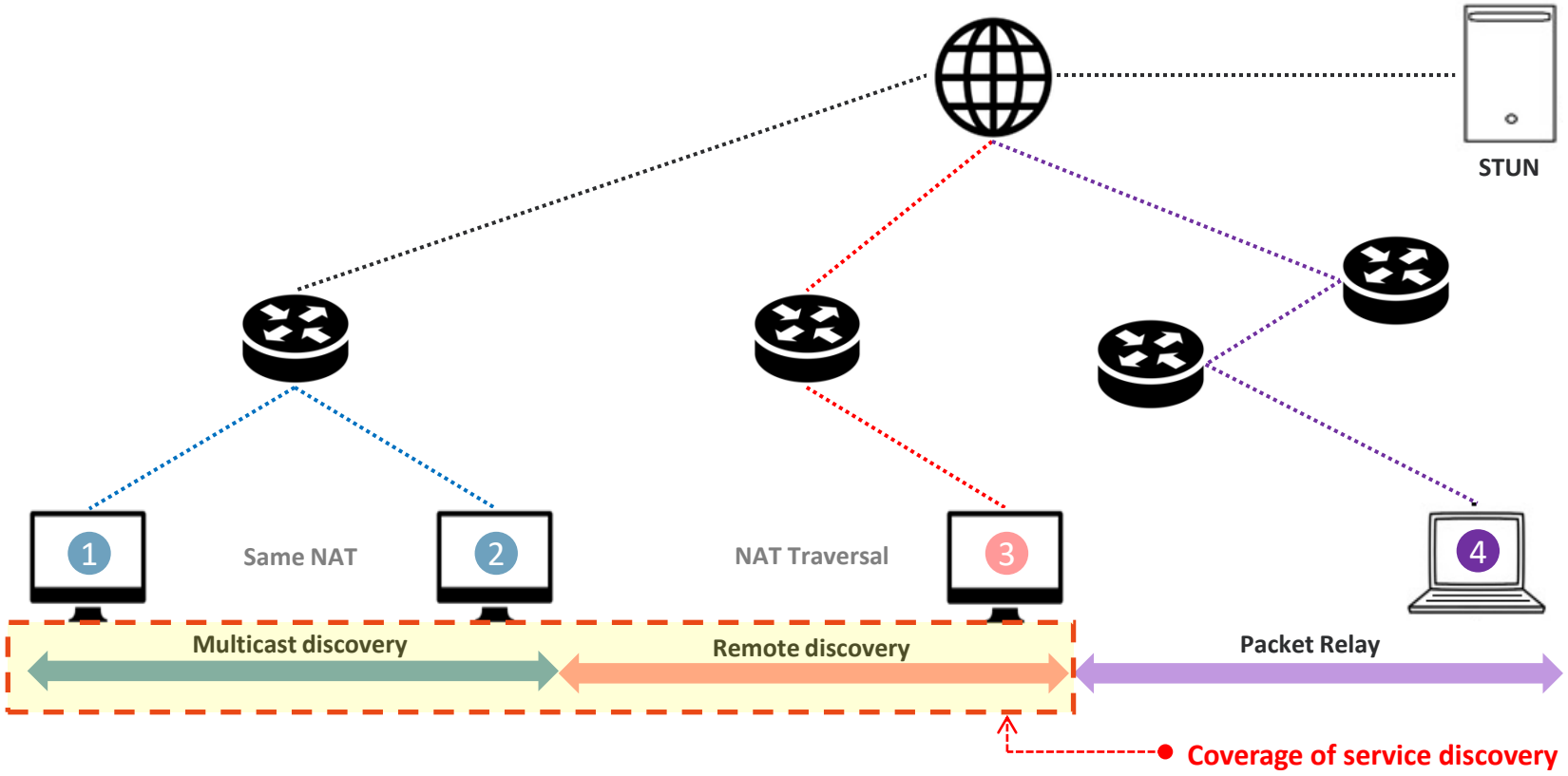
How Service Discovery Agent Works

Overview



How Service Discovery Agent Works

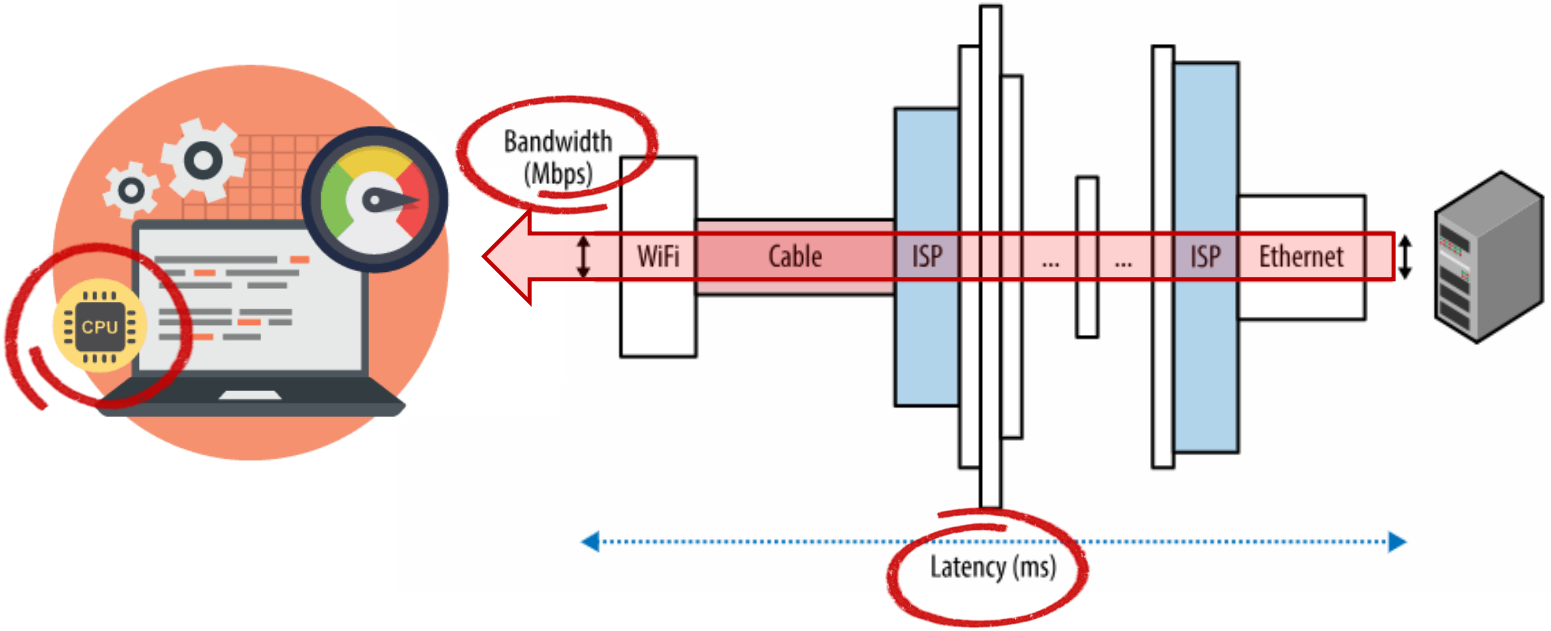
Considerable device network environment



Connection device	Discovery method	Network Env.	Description
(1 ~ 2)	Multi-Cast MDNS	Same NAT	Direct send / recv
(1, 2 ~ 3)	NAT Traversal (direct connection)	Single NAT	NAT Traversal with UDP hole punching
(1, 2, 3 ~ 4)	STUN Server	Nested NAT	Packet relay via STUN server (*Not supported)

How Service Discovery Agent Works

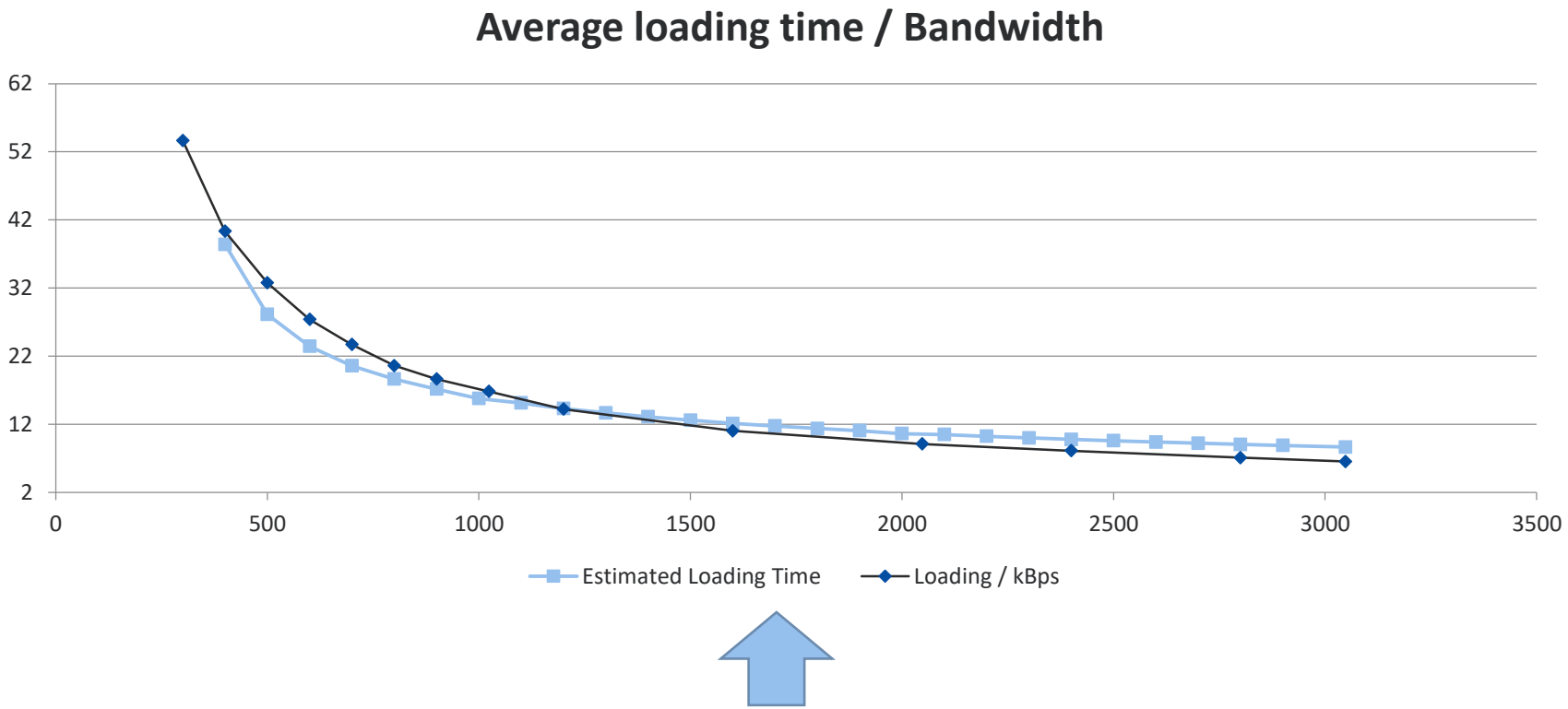
- **Offloading decision**
  - Goal : Select best-fit provider from available device list
  - Consideration
    - Best performance
    - Service stability
    - Usability consistency
  - Effective Factor
    - Resource status : Network bandwidth, CPU load and frequency
    - Network latency between service provider & consumer



How Service Discovery Agent Works

Select Best Performing Renderer

- Key factor: Network bandwidth, CPU usage

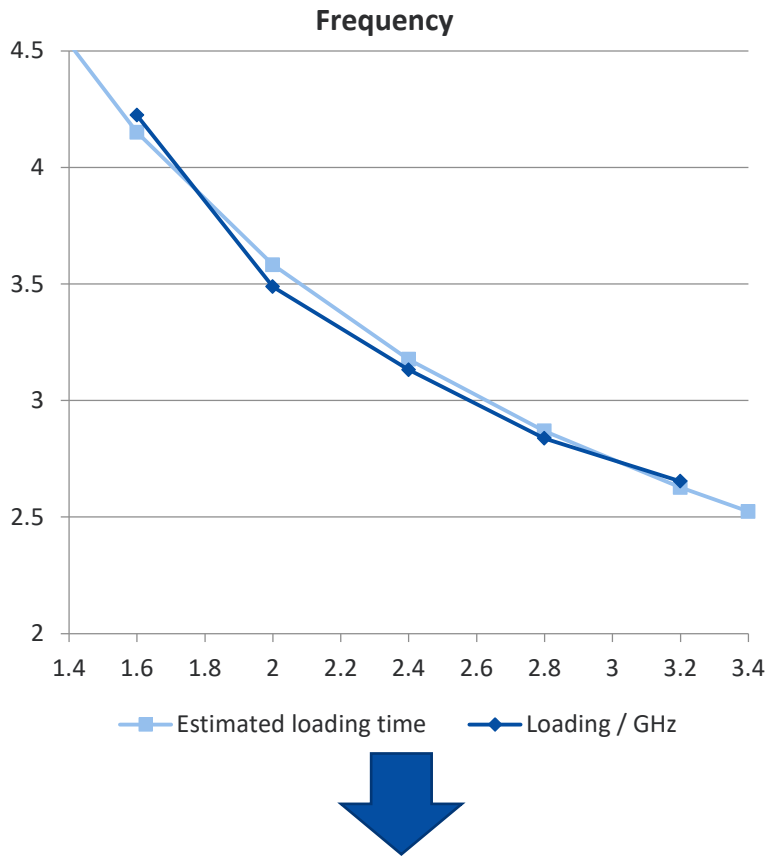


$$Loading\ Time \approx 8770 * x^{-0.9}$$

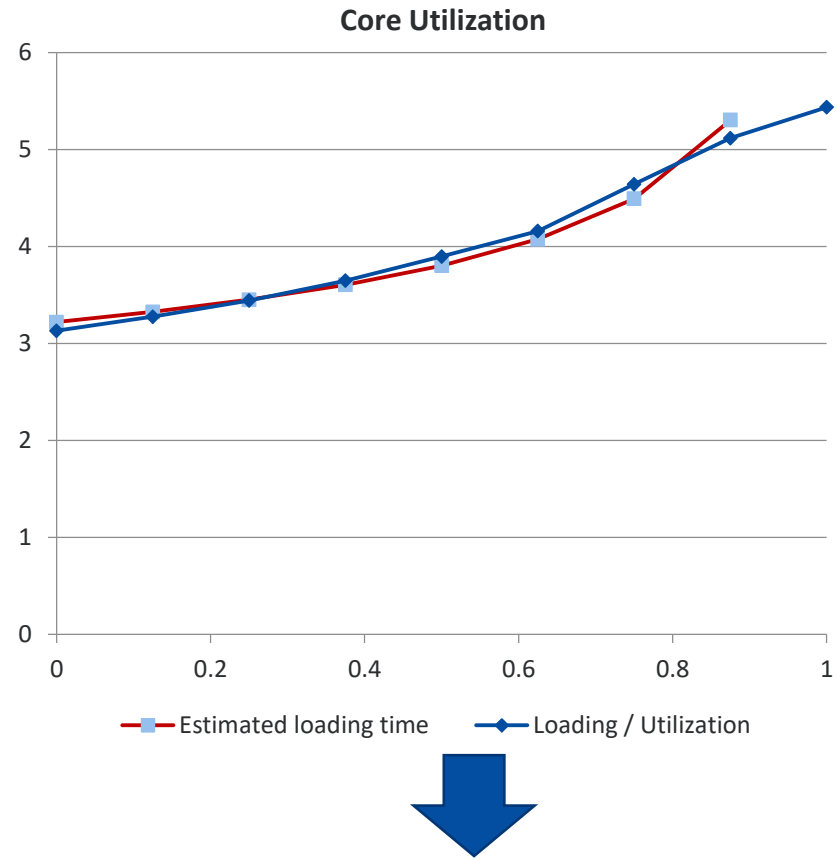
How Service Discovery Agent Works

Select Best Performing Renderer

- Key factor: Network bandwidth, CPU usage (Frequency, Utilization, # Cores)



$$Loading\ Time \approx 5.66 * x^{-0.66}$$



$$Loading\ Time \approx 3.22 * x^{-0.241}$$

x: Normalized CPU utilization

How Service Discovery Agent Works

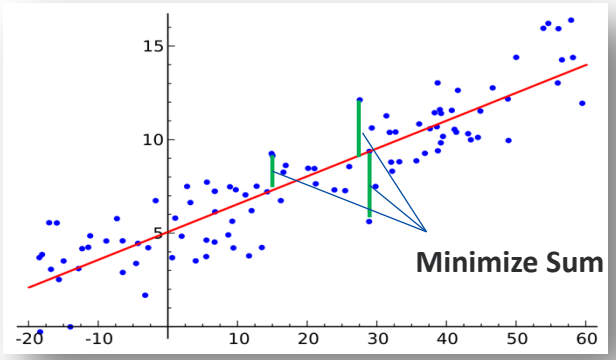
- **Renderer Score Calculation**
  - Network score + CPU score (frequency score, utilization score, core score)
  - $$Score(m) = \frac{\frac{1}{8770\, n^{-0.9}} + \frac{\frac{1}{5.66f^{-0.66}} + \frac{1}{3.22u^{-0.241}} + \frac{1}{4c^{-0.3}}}{3}}{2} + 0.77r^{-0.43}$$

    Network score
    CPU score
    Page loading score
    Rendering score

- **Select Renderer With Service Stability**
  - Moving average from renderer scores
  - $StableScore = \frac{1}{n}\sum_{i=0}^{n-1} Score(m-i)$

Linear Regression Technique

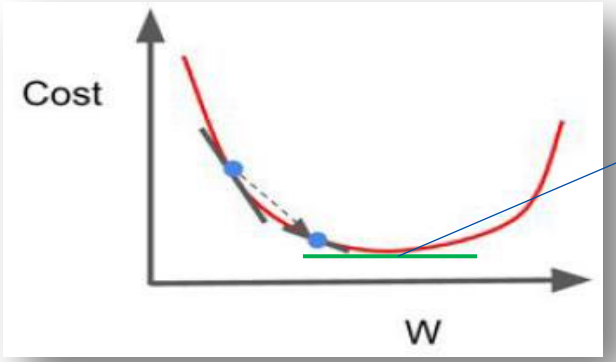
- How to find best-fit line?
  - MSE(Mean squared error): minimize distance between line and dots



$$\text{cost} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Finding minimized cost for parameter w_1 and w_2 for $y = w_1 x + w_2$

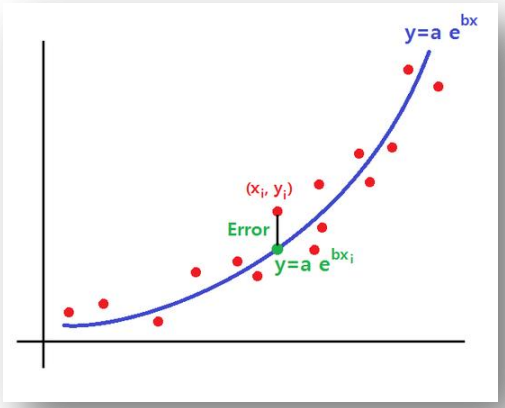
Differentiation



Find minimum cost where gradient = 0

Polynomial Regression Technique

- **What if line is not linear but polynomial?**



$$y = a\,e^{bx}$$

$$\Rightarrow \ln y = \ln(a\,e^{bx})$$

$$\Rightarrow \ln y = \ln a + \ln e^{bx}$$

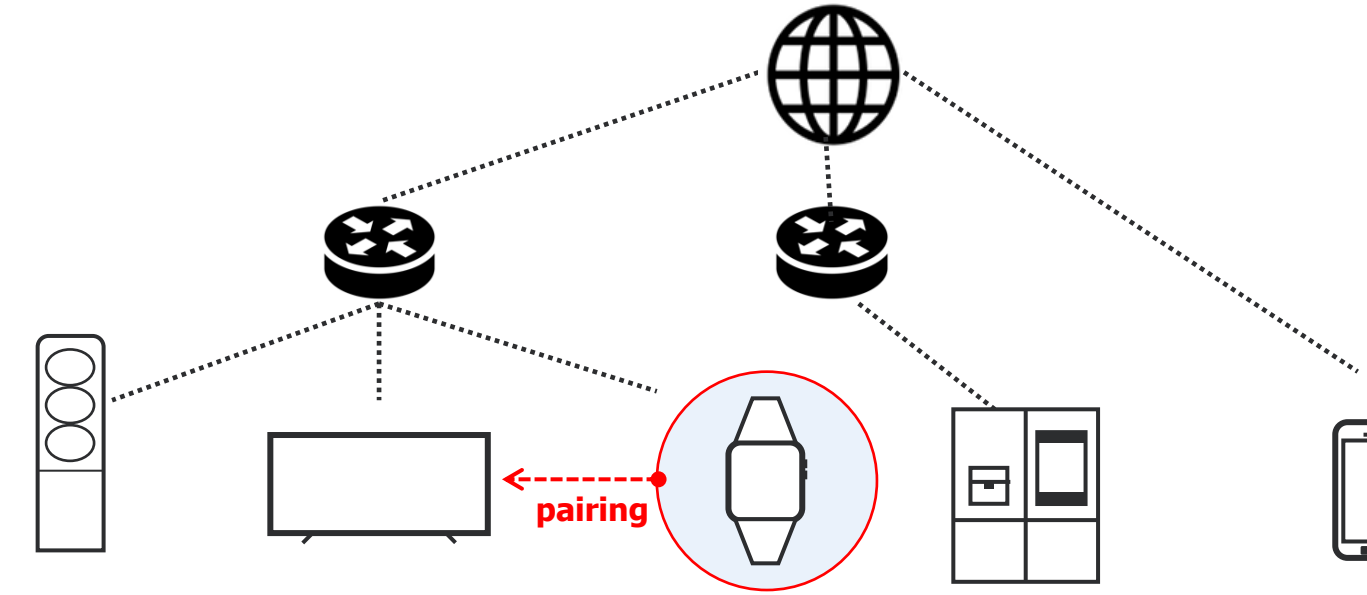
$$\Rightarrow \ln y = \ln a + bx$$

Let $\ln y = z$, $\ln a = w_2$, $b = w_1$

$$z = w_1 x + w_2$$ ⇨ **Linear regression!**

How Service Discovery Agent Works

- **Device Pairing :** Example of in-home device configuration



Fridge
- 8 Core @ 3.2GHz
- Utilization: 75%
- Bandwidth: 10MB

TV
- 4 Core @ 1.6GHz
- Utilization: 50%
- Bandwidth: 30MB

Refrigerator
- 4 Core @ 2.4GHz
- Utilization: 25%
- Bandwidth: 1MB

Smart phone
- 2 Core@ 2.0GHz
- Utilization: 0%
- Bandwidth: 2MB

Score: 2.68	Score: 3.40	Score: 2.10	Score: 2.28
7.8sec	5.96sec	17.79sec	10.79sec

Methodology

- Local system target
  - Chrome
  - Firefox
  - Opera-mini
  - Castanets BP

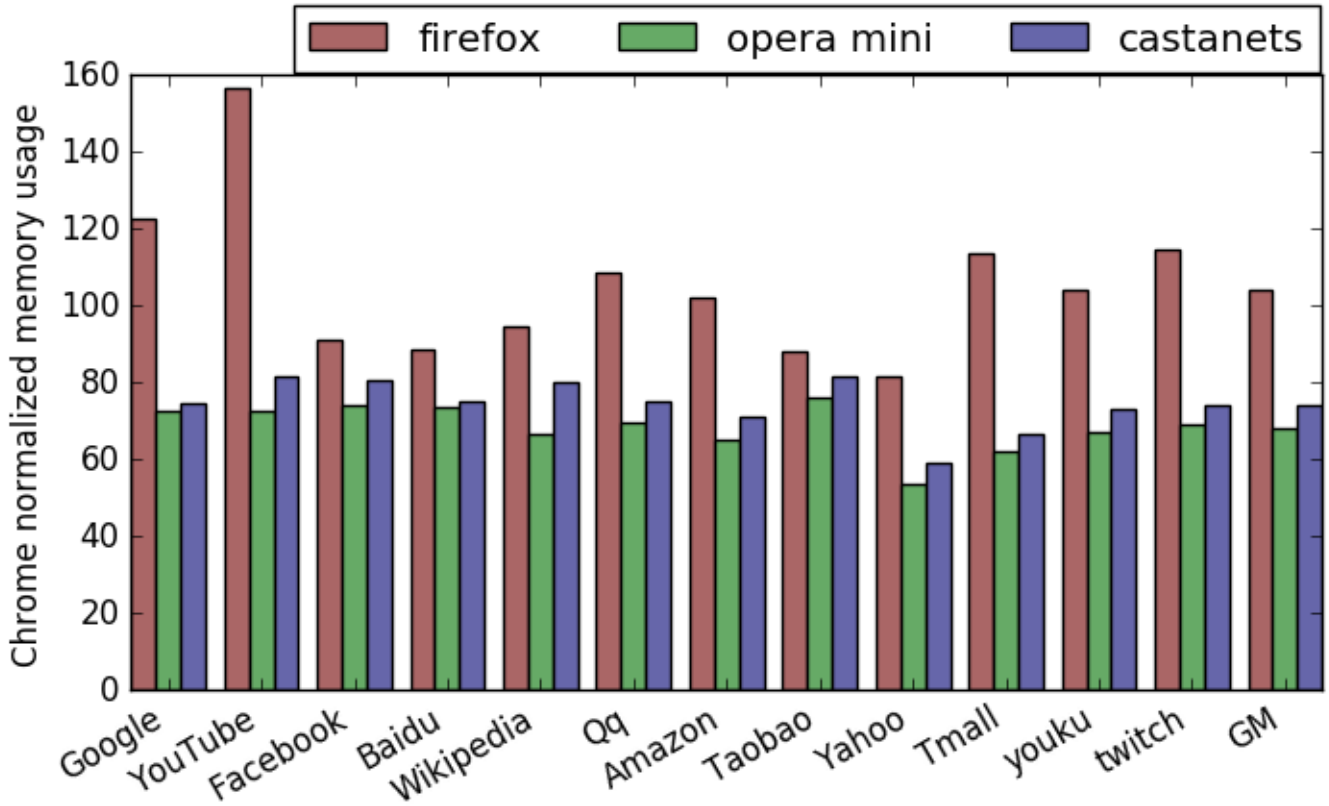
	Local Node Specification
Core	Arm Cortex A55 big-little octa-core Bigcore: 4 @ 2.7GHz Littlecore: 4 @ 1.8GHz
Memory	4GB DDR
OS	Android

- Remote system target
  - Castanets RP

Samsung QB75H

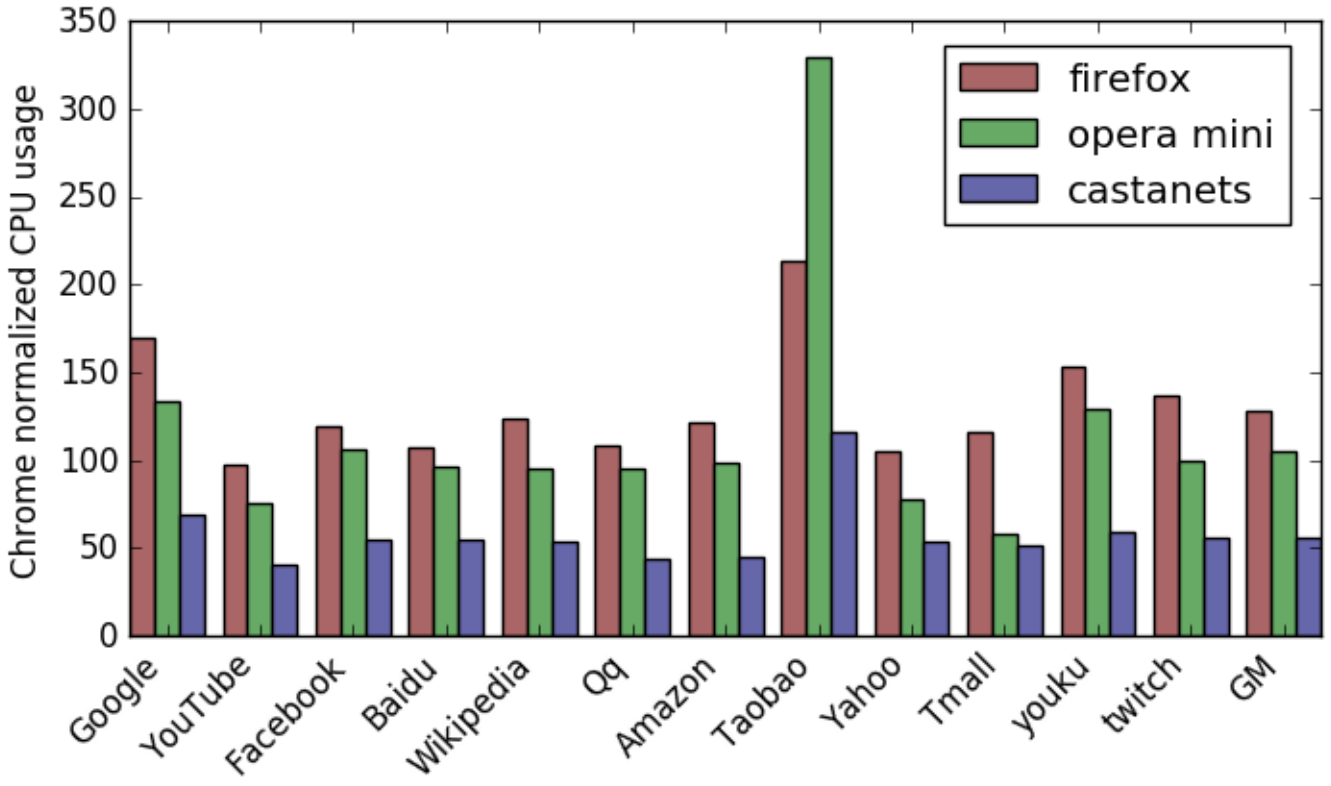
Memory Usage Evaluation

- Firefox: +3.89%
- Opera mini: -31.73%
- Castanets BP: -25.94%



CPU Utilization Evaluation

- Firefox: +27.64%
- Opera mini: +5.02%
- Castanets BP: -44.08%



Conclusion

- Castanets is an edge distributed browser
  - Browser process + renderer process + orchestration
  - Browser process: graphics operation
  - Renderer process: calculations
  - Orchestration: load balancing, pairing
- Communication problem exists in Castanets
  - Between heterogeneous web engines
  - Between different versions
    - → IPC message with different parameter
- Short way to solve problems
  - Orchestration engine can manage versions and engine type
- Long way to solve problems
  - Match all communication messages between browsers