

Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?

The Turtles Project: Design and
Implementation of Nested Virtualization

Deepak Kumar



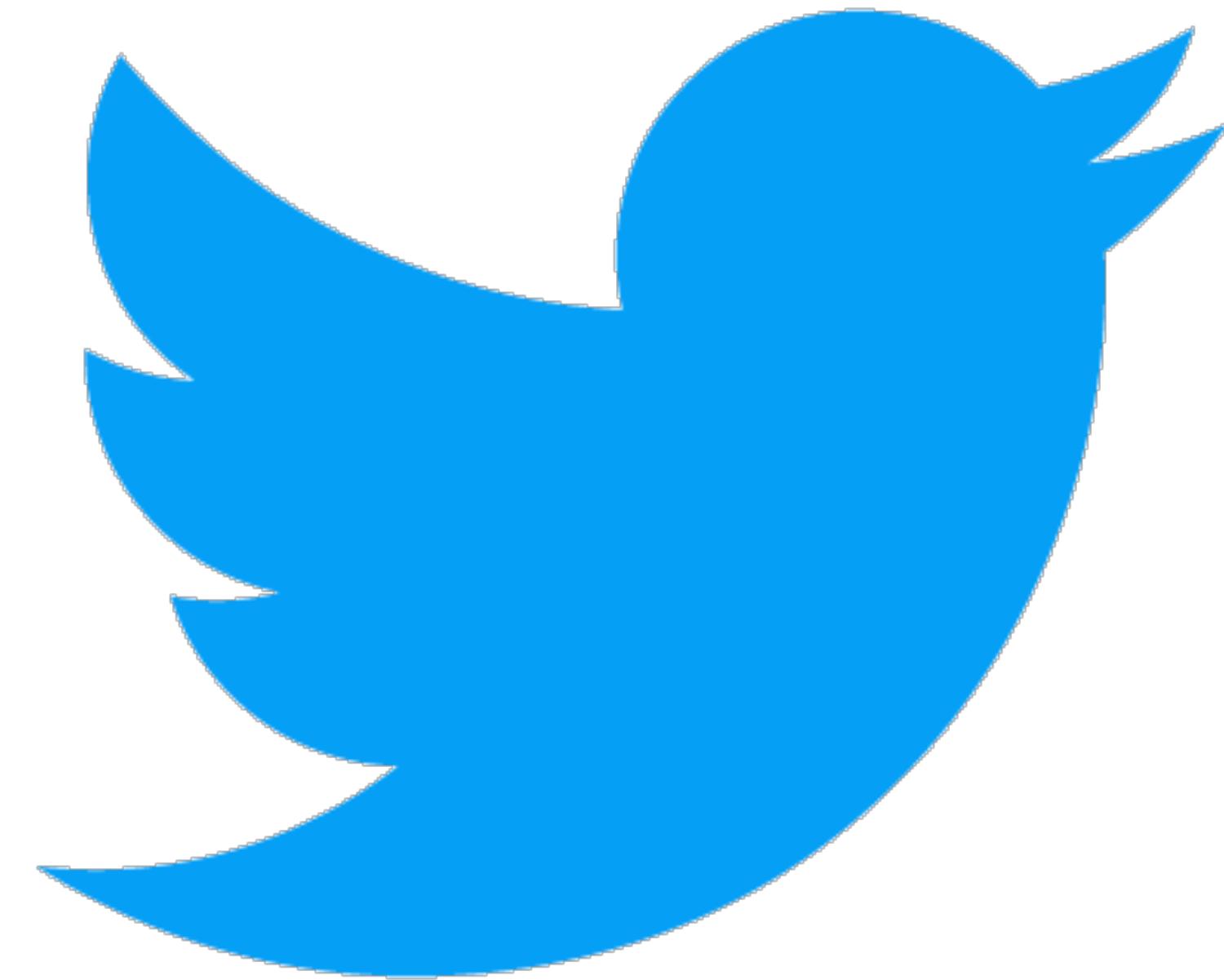
Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?

Zi Chu, Steven Gianvecchio, Haining Wang, Sushil Jajodia
IEEE Transactions in Dependable and Secure Computing



Twitter

- Online social networking and microblogging tool



Twitter

- Online social networking and microblogging tool
- Users post “tweets” - short, 280 character messages



Twitter

- Online social networking and microblogging tool
- Users post “tweets” - short, 280 character messages
- Users can be *followers* or *friends* of other users



Twitter

- Online social networking and microblogging tool
- Users post “tweets” - short, 280 character messages
- Users can be *followers* or *friends* of other users
- By end of Q4 2017, Twitter had ~330M monthly active users^[1]



Automation on Twitter

- Automated programs, or **bots**, are prevalent on Twitter
 - In fact, integral to Twitter's design
 - “News bots”



Automation on Twitter

- Automated programs, or **bots**, are prevalent on Twitter
 - In fact, integral to Twitter's design
 - “News bots”
 - However, bots can also be used in *malicious* ways



RUSSIAN BOTS ARE USING 2016 TACTICS TO HIJACK THE GUN DEBATE ON TWITTER

ERIN GRIFFITH SECURITY 02.15.18 02:00 PM

PRO-GUN RUSSIAN BOTS FLOOD TWITTER AFTER PARKLAND SHOOTING

After Florida School Shooting, Russian ‘Bot’ Army Pounced

By SHEERA FRENKEL and DAISUKE WAKABAYASHI FEB. 19, 2018



How can we determine if a Twitter user is automated?



Collecting Users + Tweets

Twitter Crawl

Timeline API



Collecting Users + Tweets

Twitter Crawl

Timeline API



Crawling Twitter

- Crawl Twitter with a Depth-First-Search



Crawling Twitter

- Crawl Twitter with a Depth-First-Search
 - Select 5 users at random as a “seed”



Crawling Twitter

- Crawl Twitter with a Depth-First-Search
 - Select 5 users at random as a “seed”
 - For each user, record their tweets and their followers



Crawling Twitter

- Crawl Twitter with a Depth-First-Search
 - Select 5 users at random as a “seed”
 - For each user, record their tweets and their followers
 - Repeat for each follower...



Crawling Twitter

- Crawl Twitter with a Depth-First-Search
 - Select 5 users at random as a “seed”
 - For each user, record their tweets and their followers
 - Repeat for each follower...
 - In total, crawled **429,423** users



Collecting Users + Tweets

Twitter Crawl

Timeline API



Timeline API

- Twitter publishes the 20 most recent Tweets in global scope
 - Collected tweets and users during same timeframe as Twitter crawl
 - Added an additional **89,984** users



Twitter Dataset

Data Volume

Metadata Collected?

Users

512,407

username, followers,
following, registration
date, verified

Tweets

41,995,545

user, time, device



Types of Users on Twitter



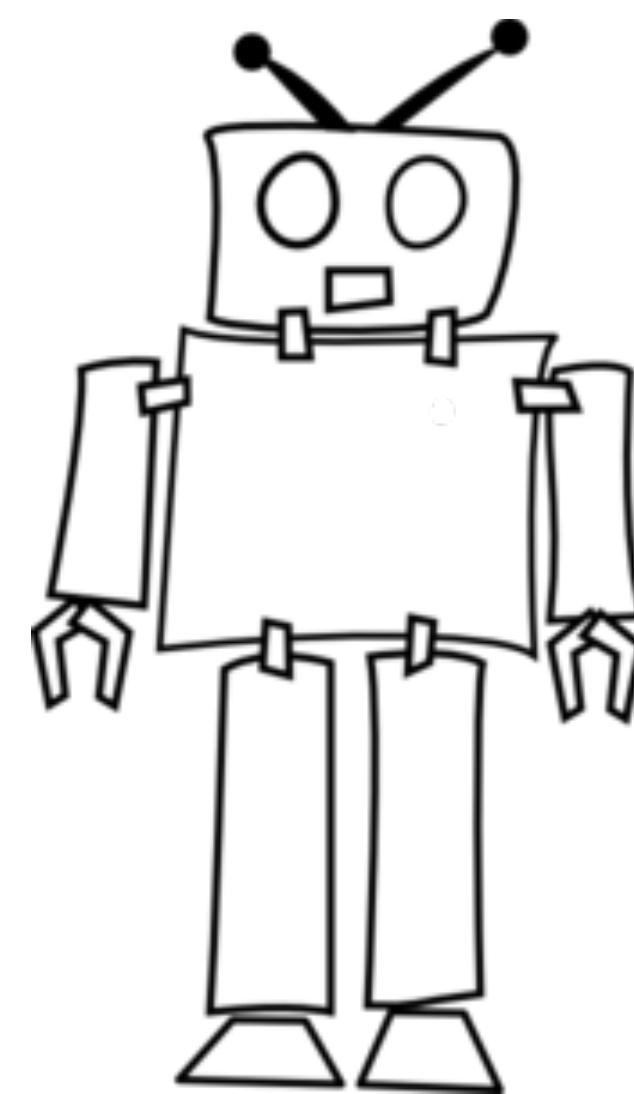
Human



Types of Users on Twitter



Human

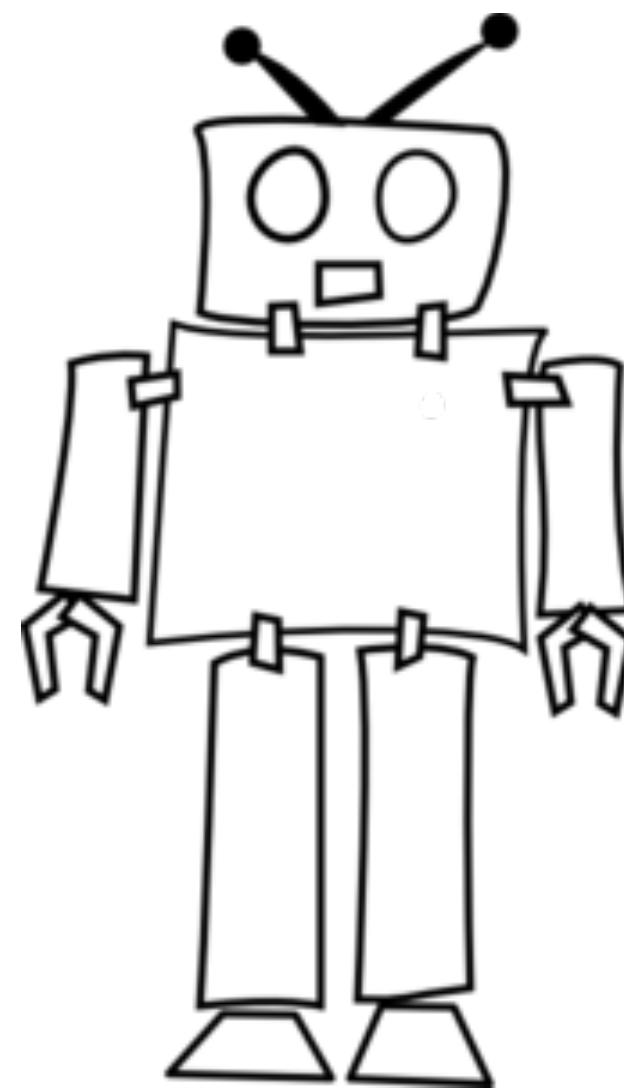


Bot

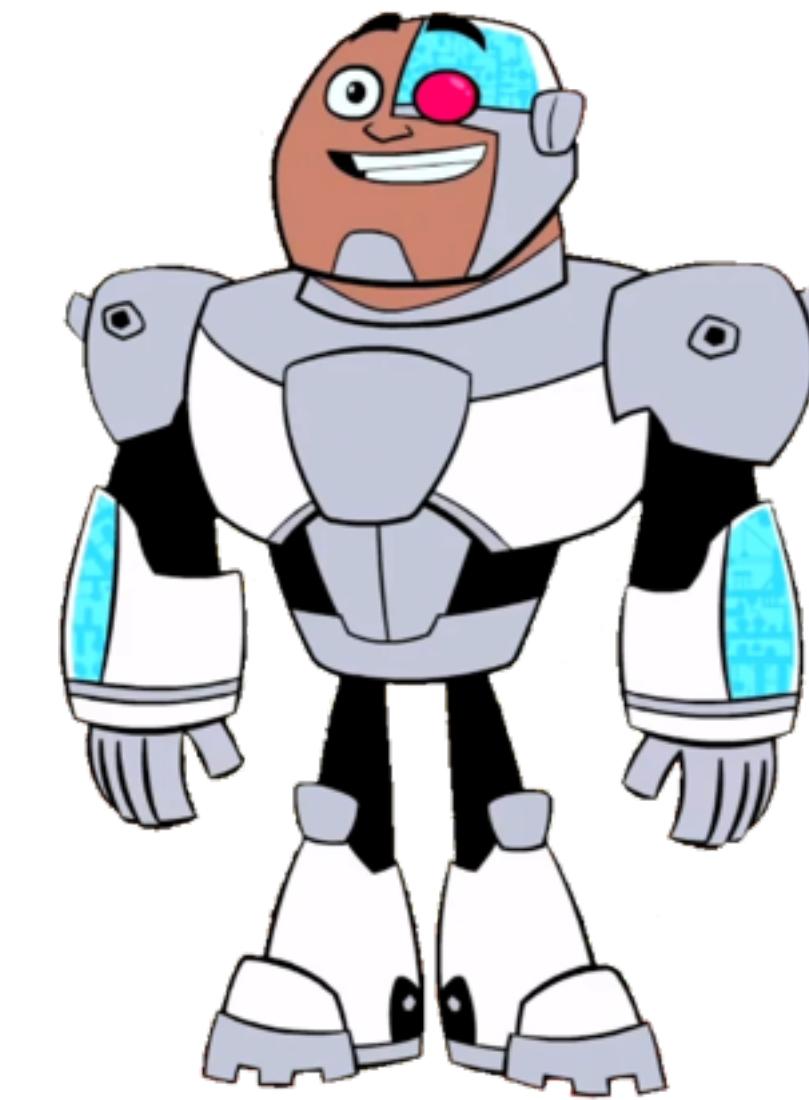
Types of Users on Twitter



Human



Bot



Cyborg

Classifying Users on Twitter: Ground Truth

- Select random sample of Twitter users from dataset
- Manually classify each user based on inspecting their profile
 - Follows the principle of “Turing Test” - 5 minutes to determine what class of user



Classifying Users on Twitter: Ground Truth

- Collected **2,000** unique users in each class
- Contributed **8,350,095** tweets



Classifying Users on Twitter: Ground Truth

- Collected **2,000** unique users in each class
- Contributed **8,350,095** tweets
- We can now investigate the *differences in behavior between user-classes*



Hypothesis: Each class of users exhibits unique behavior



Behavioral Patterns

Social Relationships

Tweeting Habits

Tweet Properties



Behavioral Patterns

Social Relationships

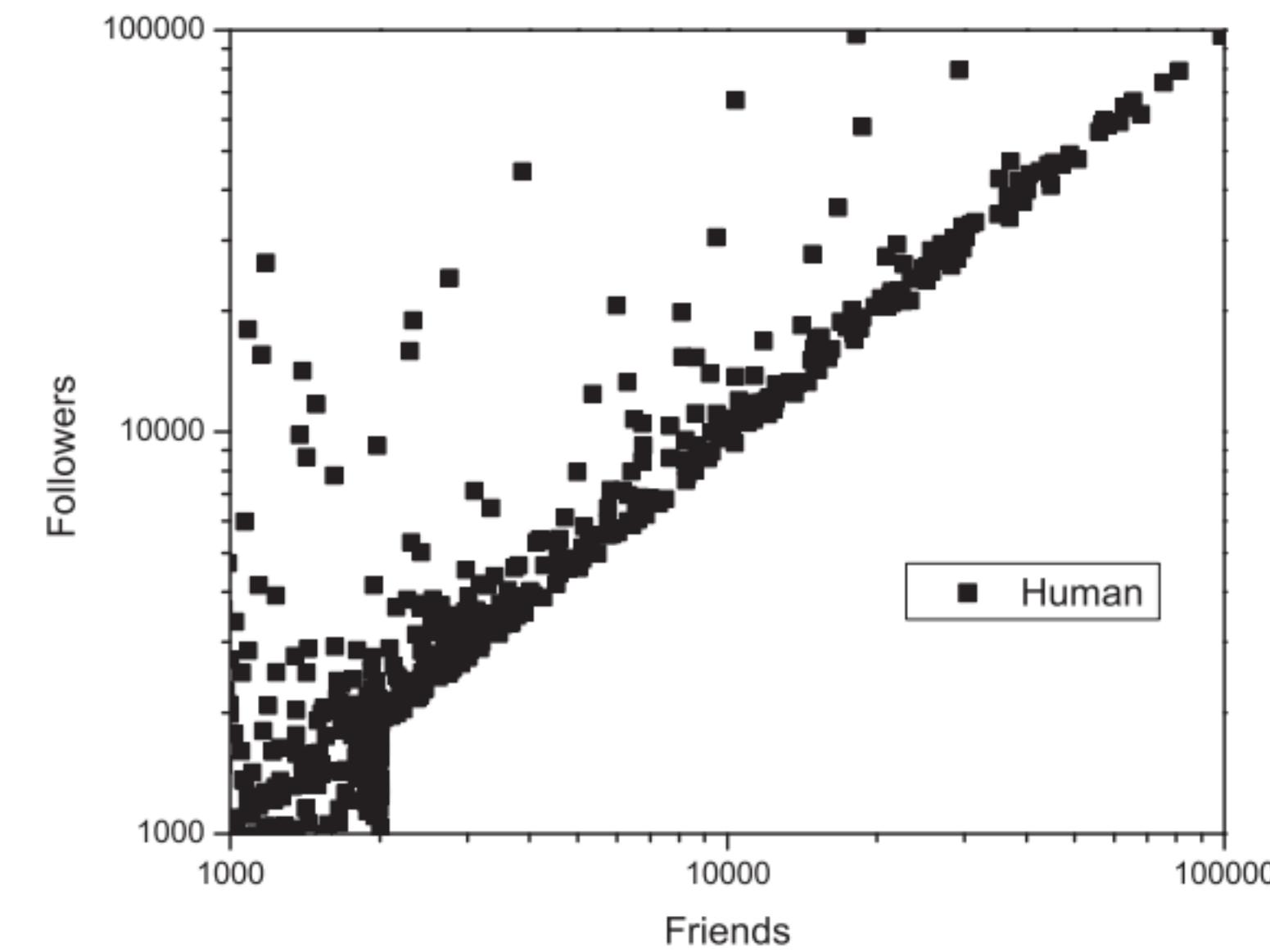
Tweeting Habits

Tweet Properties



Social Relationships: Humans

- Typically *reciprocal* relationships

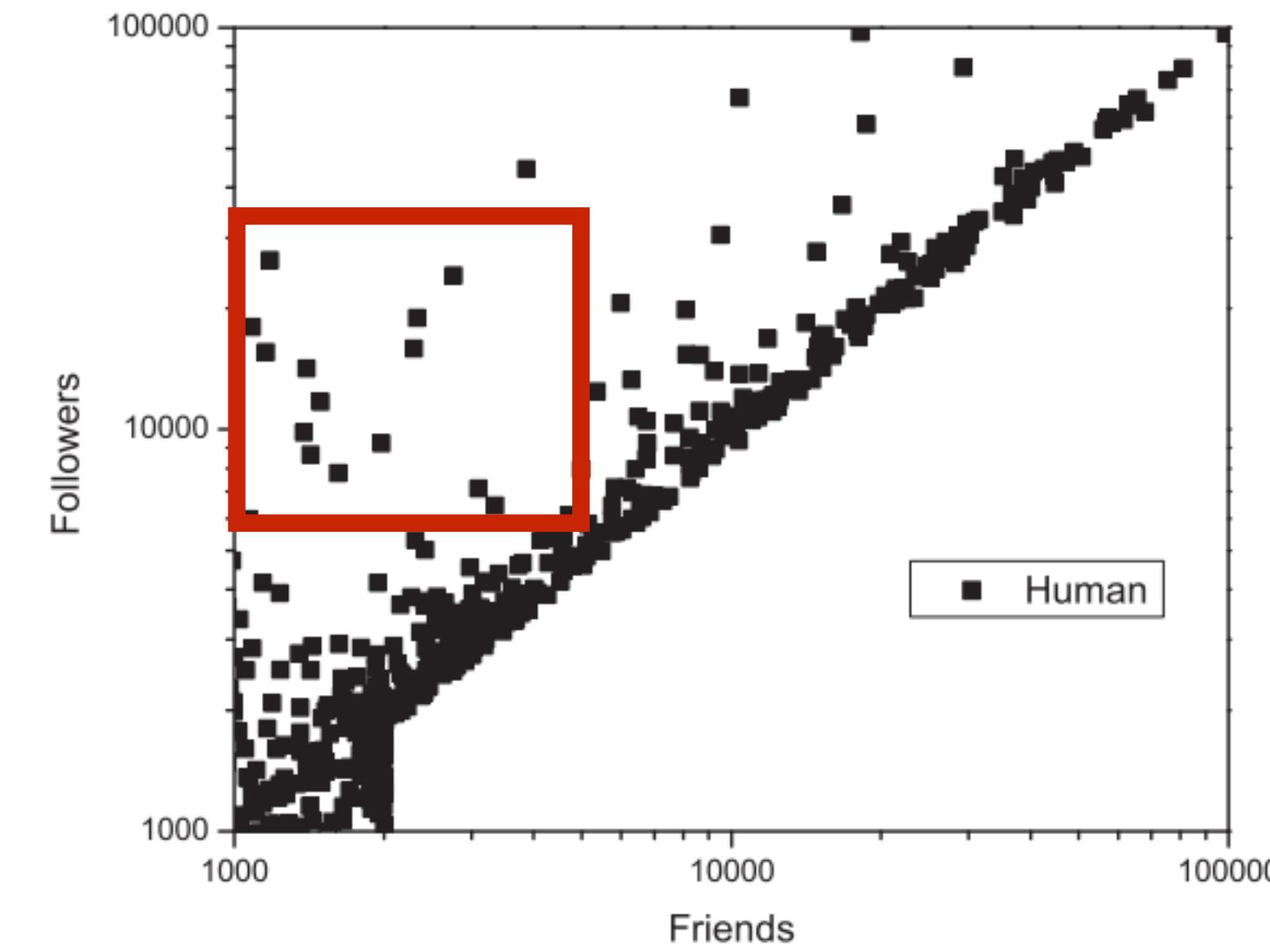


(a) Human



Social Relationships: Humans

- Typically *reciprocal* relationships
- Handful of high follower/low friends, these are mainly celebrities and important human figures

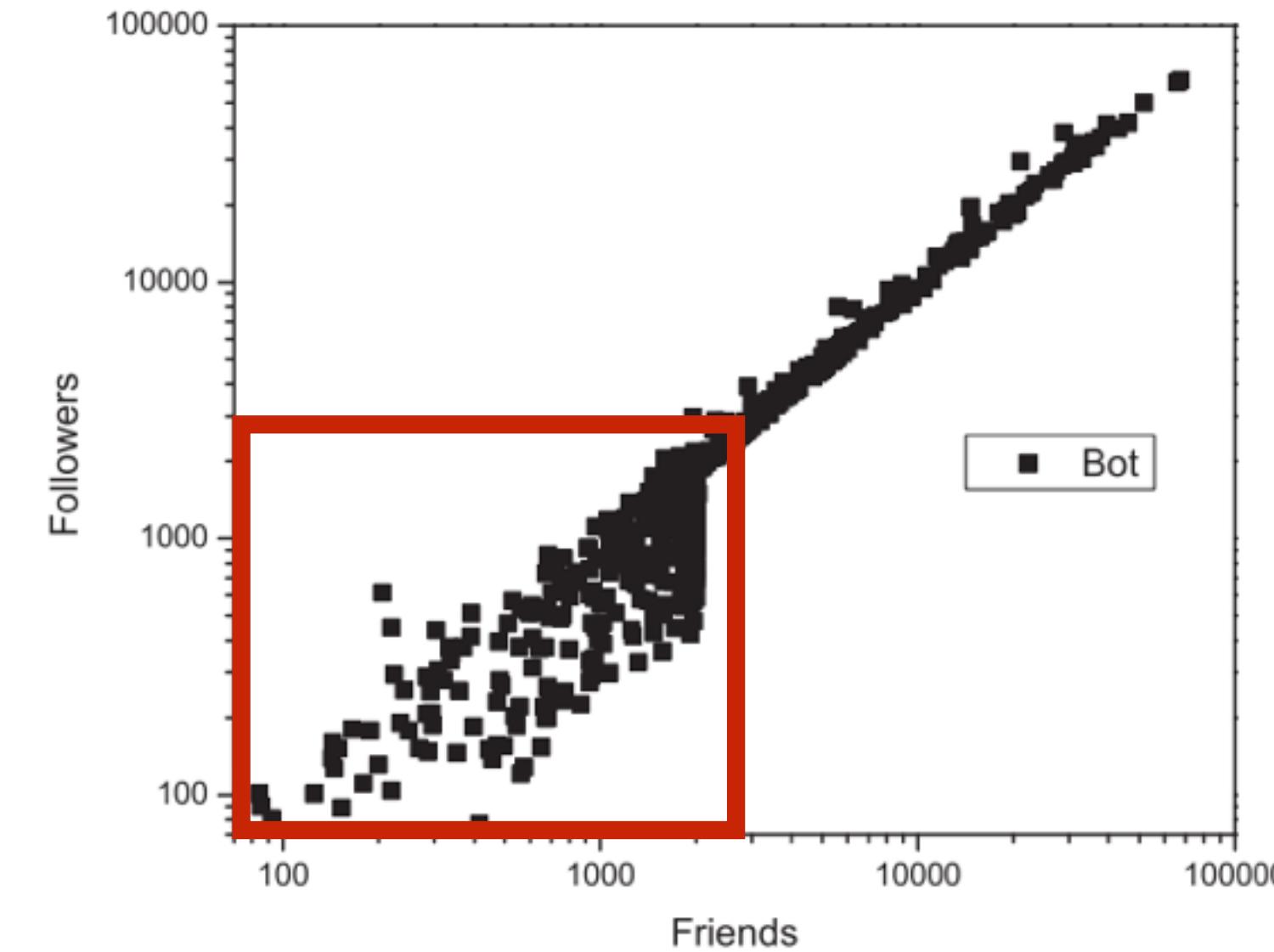


(a) Human



Social Relationships: Bots

- Most bots have lots of friends (i.e., they follow many other accounts), but not followers

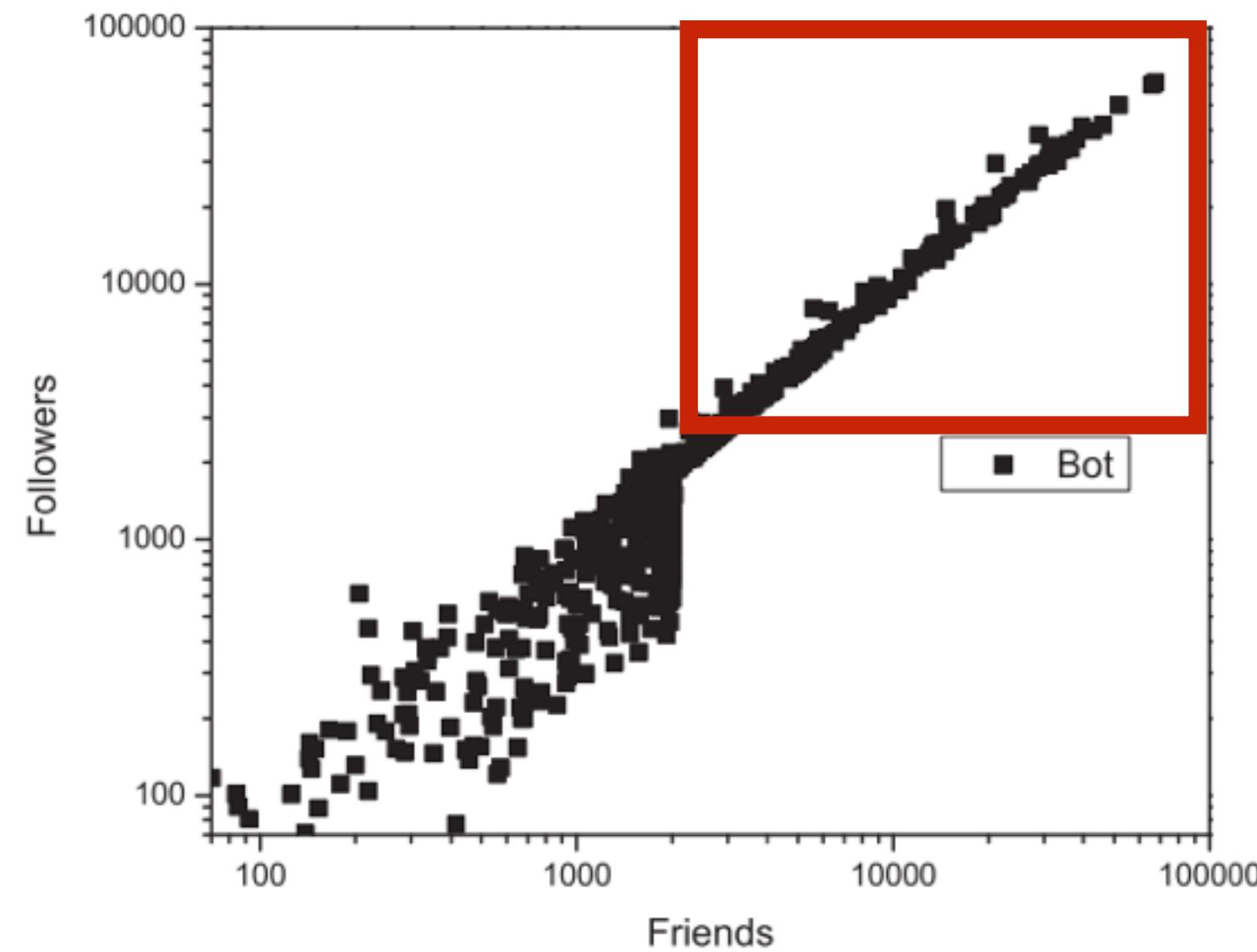


(b) Bot



Social Relationships: Bots

- Most bots have lots of friends (i.e., they follow many other accounts), but not followers
- Some bots are “smart” - they continually unfollow and follow to maintain a 1:1 ratio

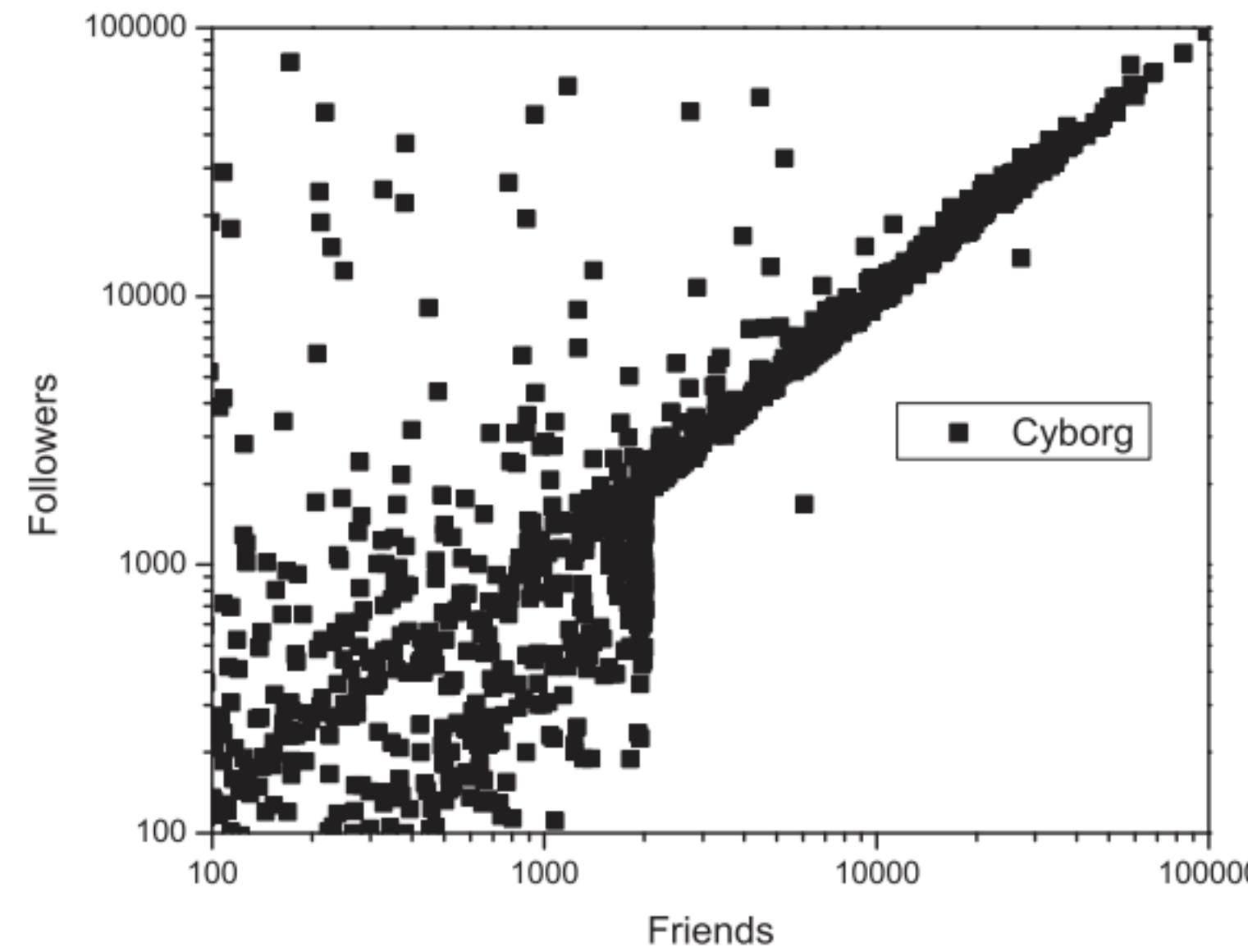


(b) Bot



Social Relationships: Cyborgs

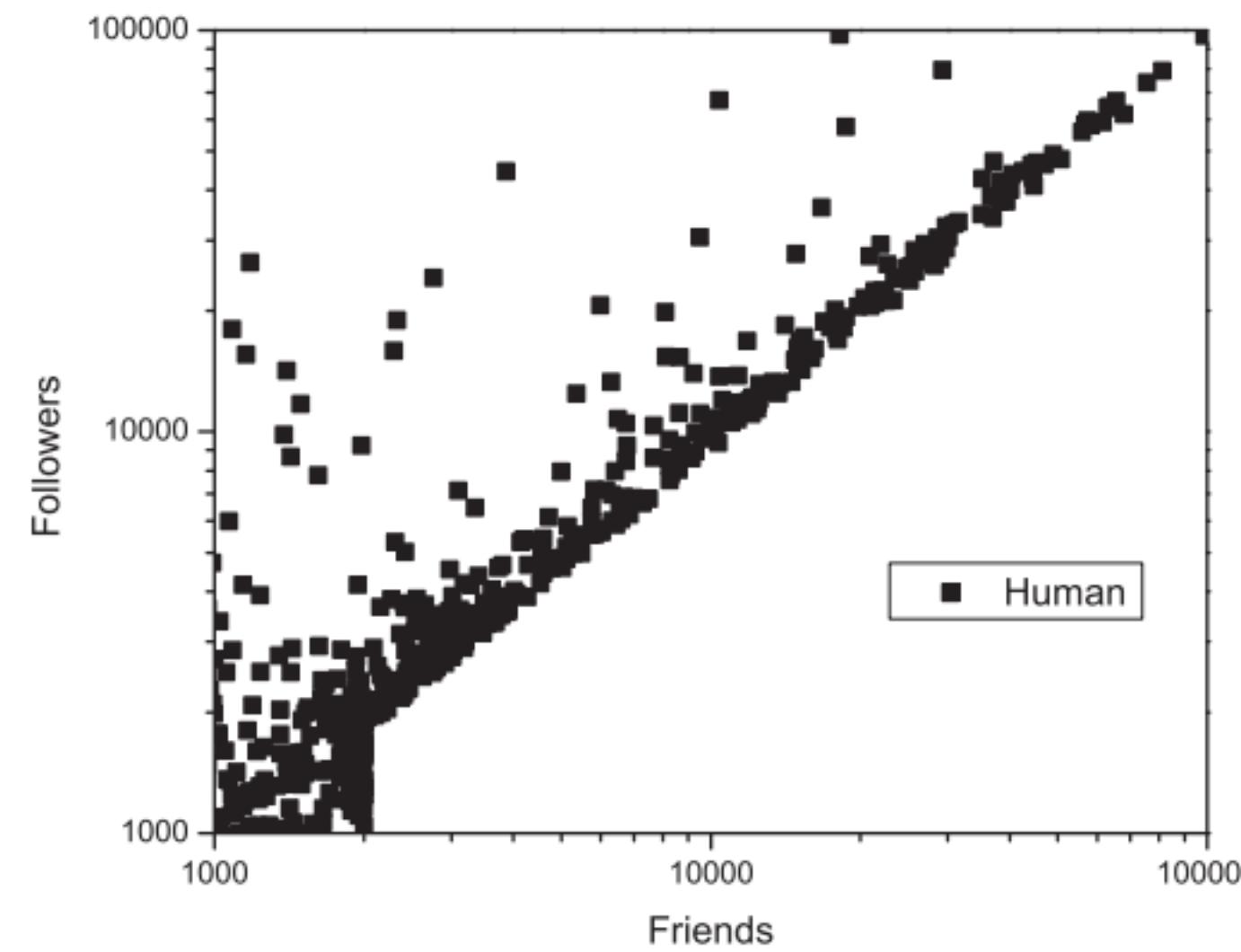
- Cyborgs exhibit a mix of human and bot social relationships



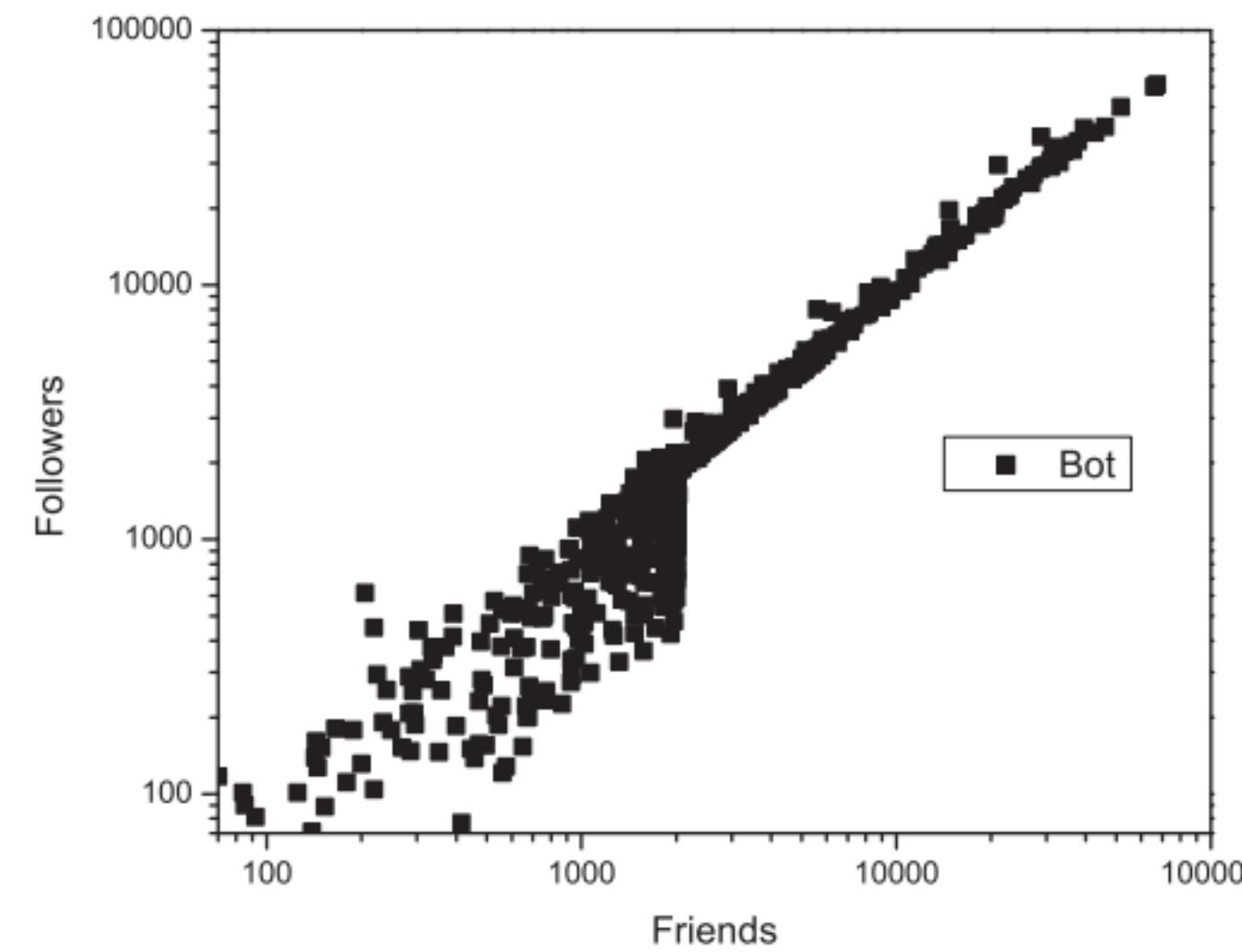
(c) Cyborg



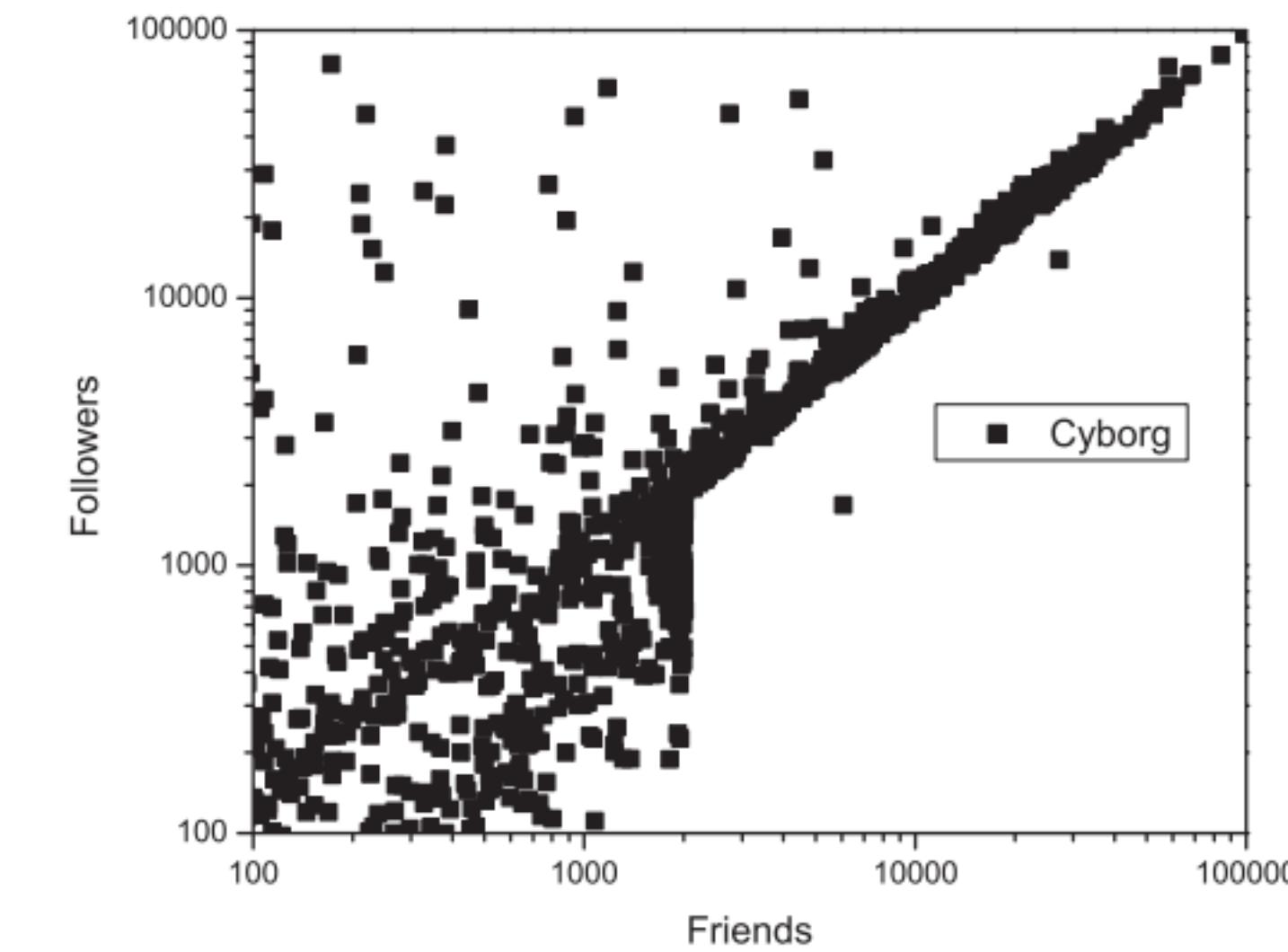
Social Relationships



(a) Human



(b) Bot



(c) Cyborg

Takeaway: Automated users follow more users than those that follow them



Behavioral Patterns

Social Relationships

Tweeting Habits

Tweet Properties



Tweeting Habits: Entropy of Tweet Timing

- Entropy is the measure of *complexity* or *chaos* of a process
- In this context, the “process” is the *timing* of tweets



Tweeting Habits: Entropy of Tweet Timing

- Entropy is the measure of *complexity* or *chaos* of a process
- In this context, the “process” is the *timing* of tweets

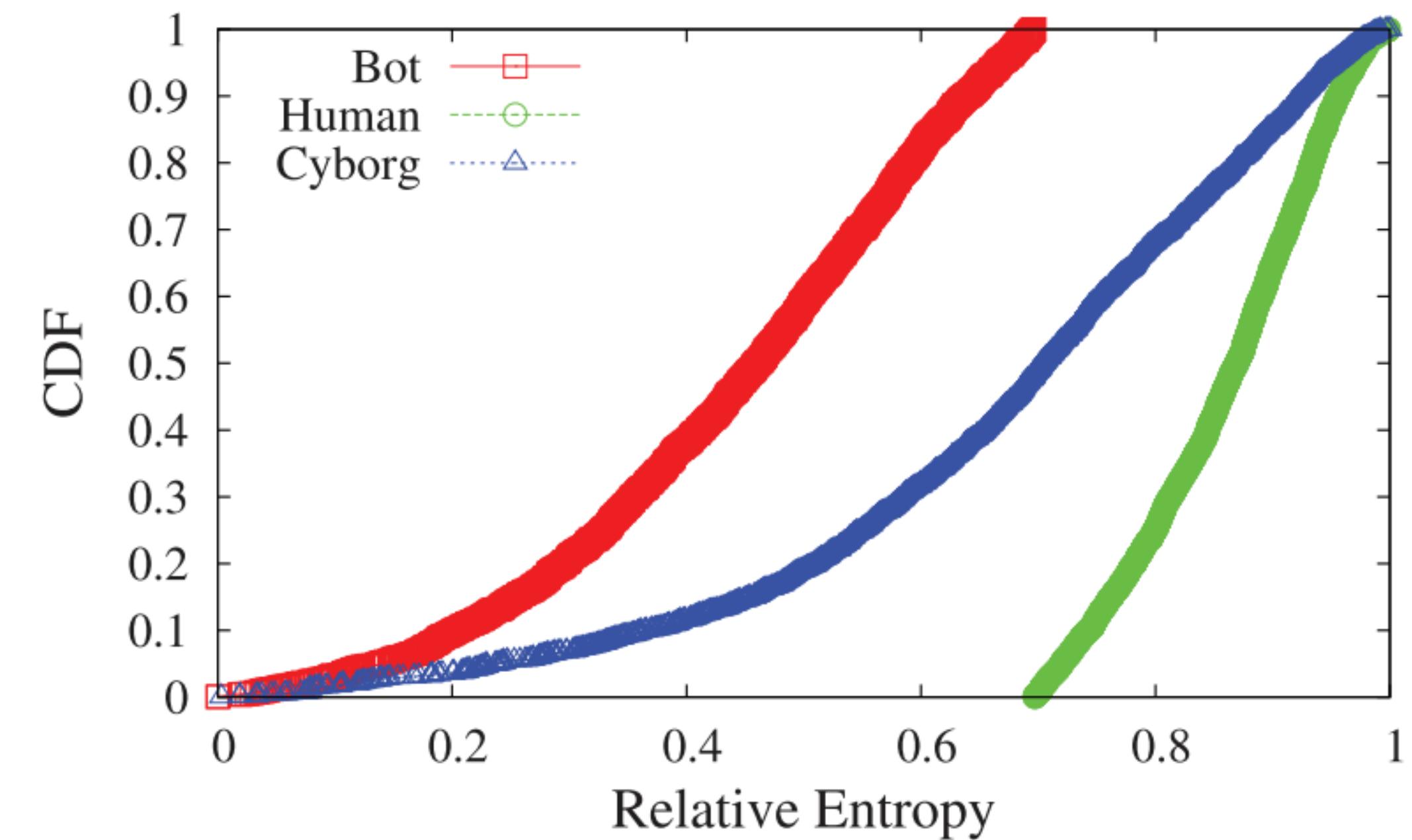
Key intuition: Automated accounts tweet at regular intervals, while humans do not



Tweeting Habits: Entropy of Tweet Timing

$$\frac{entropy(u_i)}{max(entropy(u_1), entropy(u_2), \dots, entropy(u_n))}$$

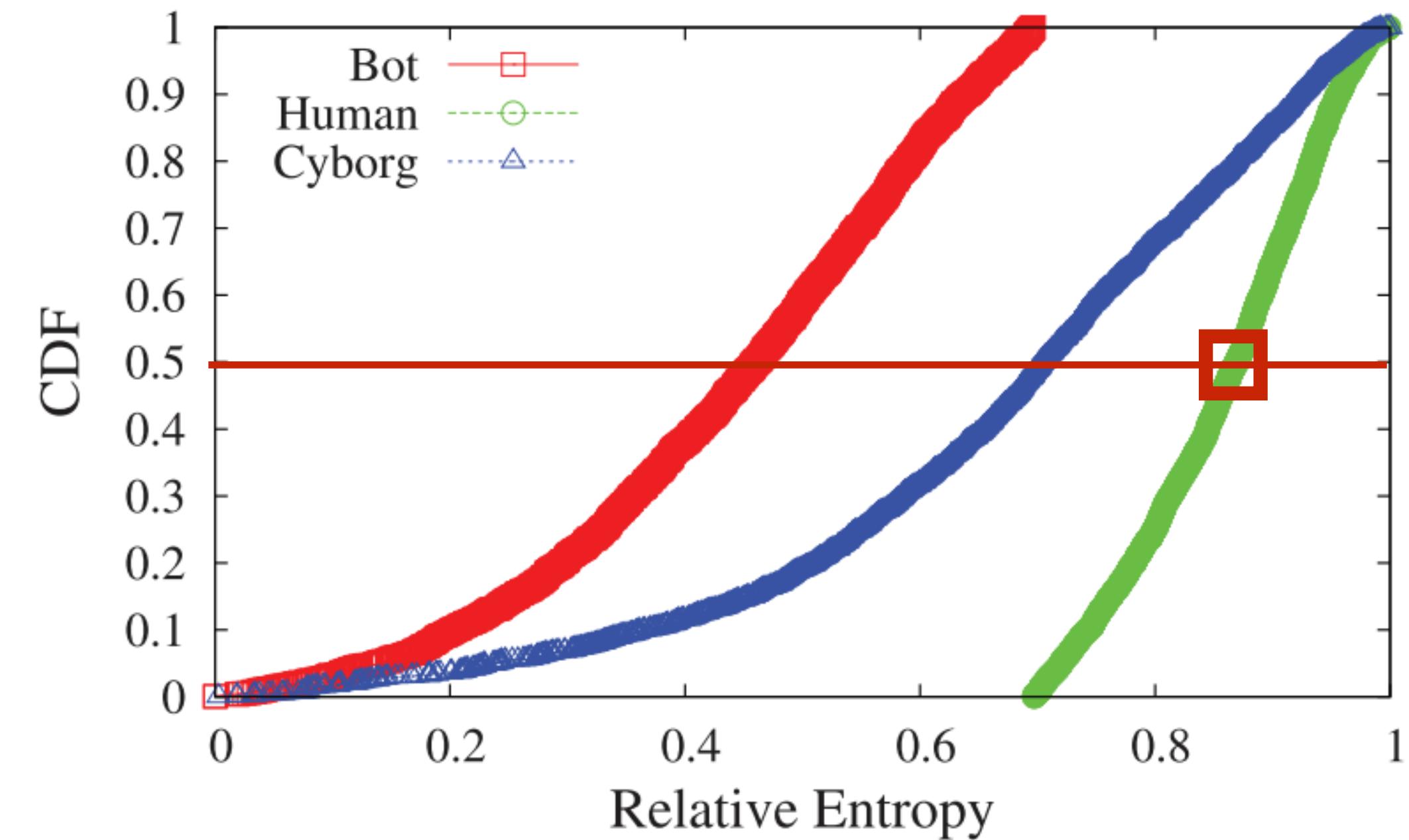
- Normalize entropy of each account by largest entropy in the dataset



Tweeting Habits: Entropy of Tweet Timing

$$\frac{entropy(u_i)}{\max(entropy(u_1), entropy(u_2), \dots, entropy(u_n))}$$

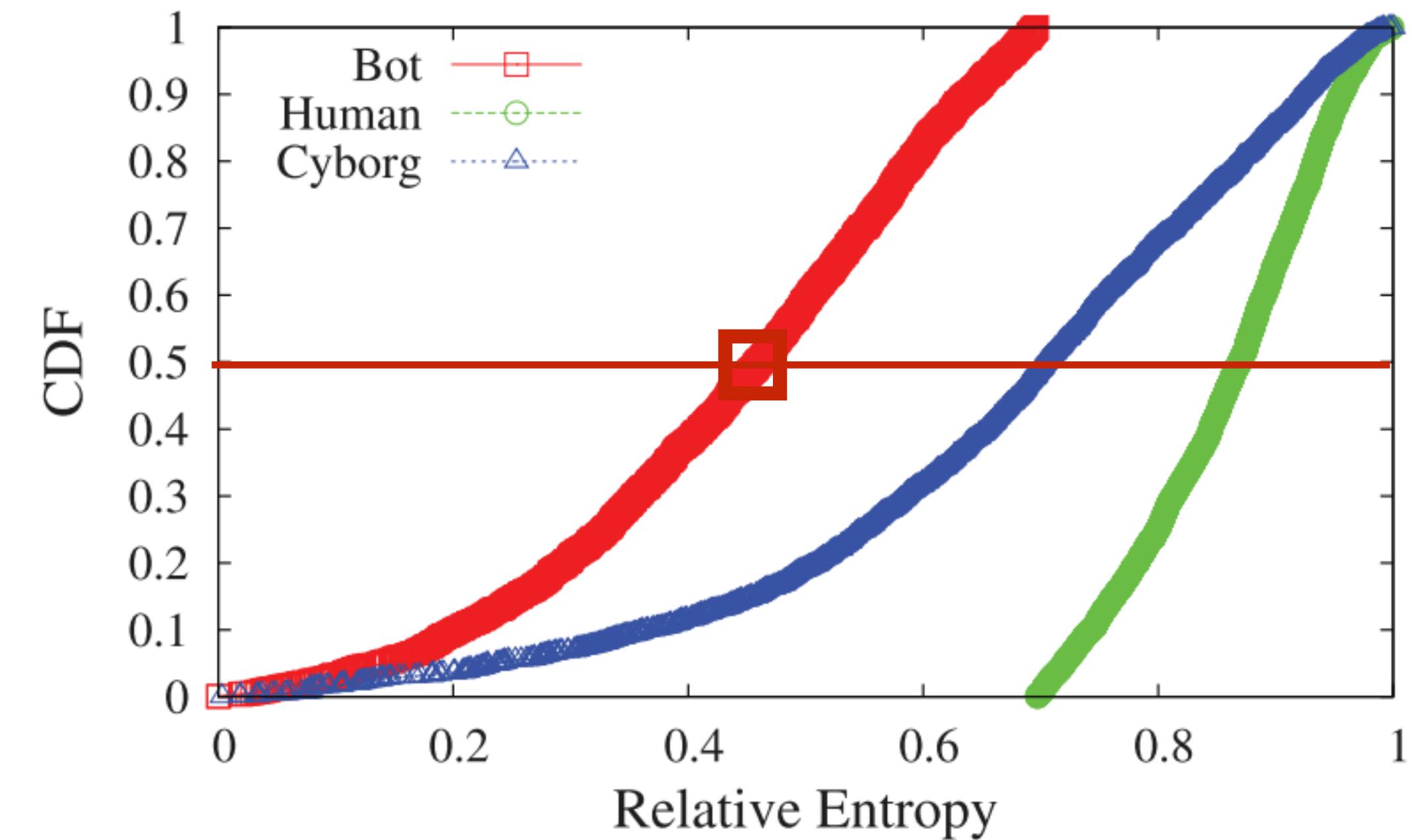
- Normalize entropy of each account by largest entropy in the dataset
- Humans exhibit highest entropy - median rel. entropy is 0.9



Tweeting Habits: Entropy of Tweet Timing

$$\frac{entropy(u_i)}{\max(entropy(u_1), entropy(u_2), \dots, entropy(u_n))}$$

- Normalize entropy of each account by largest entropy in the dataset
- Humans exhibit highest entropy - median rel. entropy is 0.9
- Bots exhibit lowest entropy - median rel. entropy is 0.4

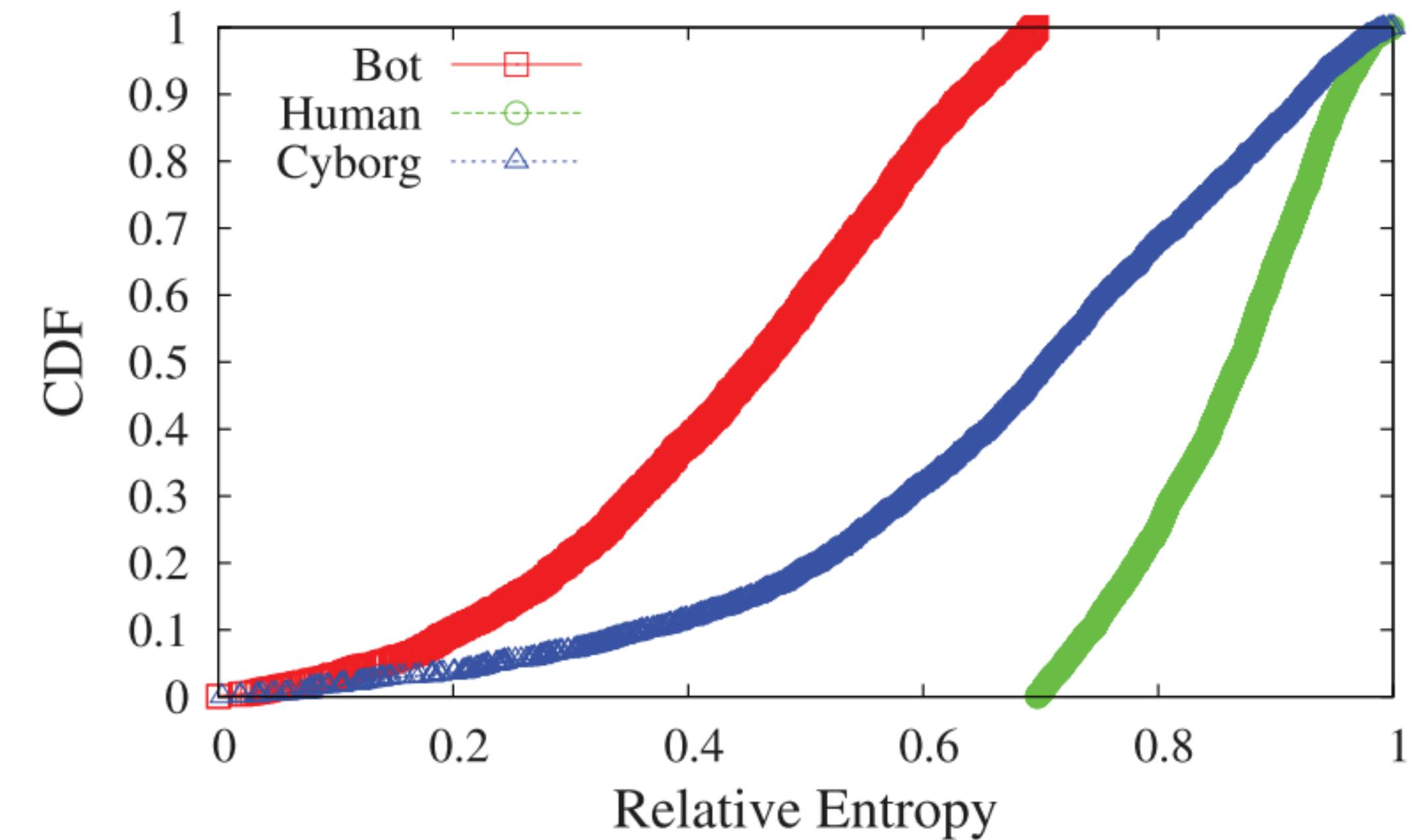


Tweeting Habits: Entropy of Tweet Timing

$$\frac{entropy(u_i)}{\max(entropy(u_1), entropy(u_2), \dots, entropy(u_n))}$$

- Normalize entropy of each account by largest entropy in the dataset
- Humans exhibit highest entropy - median rel. entropy is 0.9
- Bots exhibit lowest entropy - median rel. entropy is 0.4

Takeaway: Human tweeting behavior is more random than bot tweeting behavior



Tweeting Habits: What tools are used to Tweet?

- Users can Tweet via a wide-array of channels
 - Web
 - Mobile
 - Third party applications
 - APIs

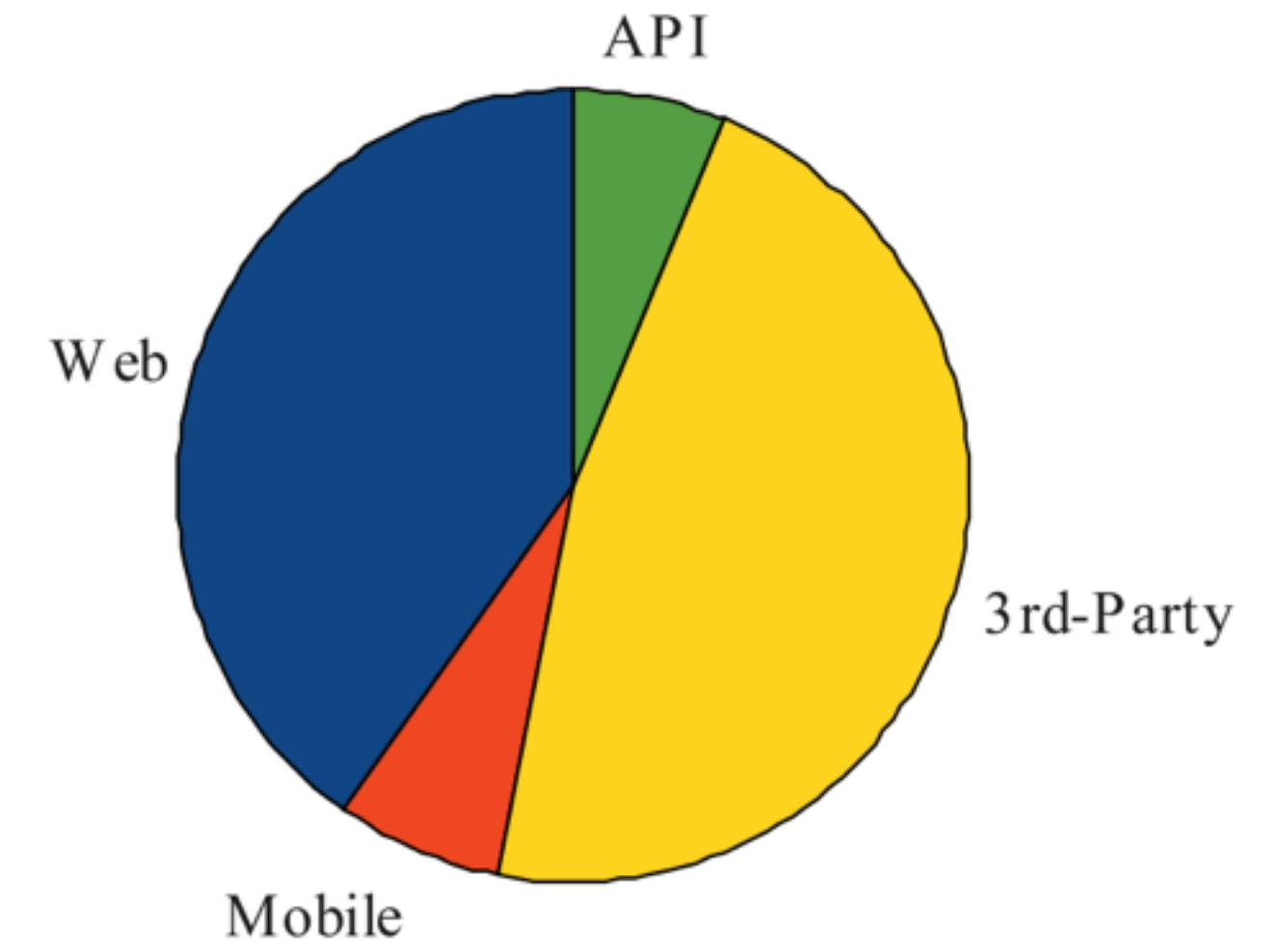
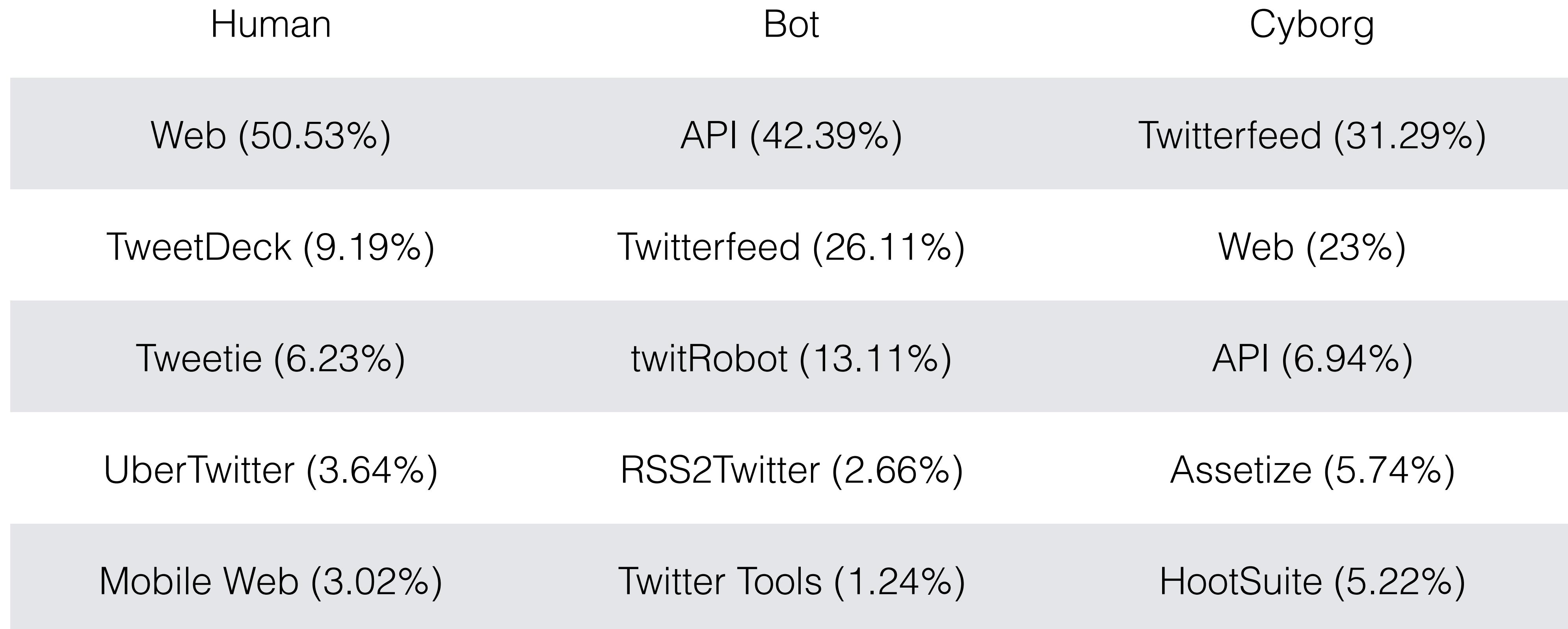


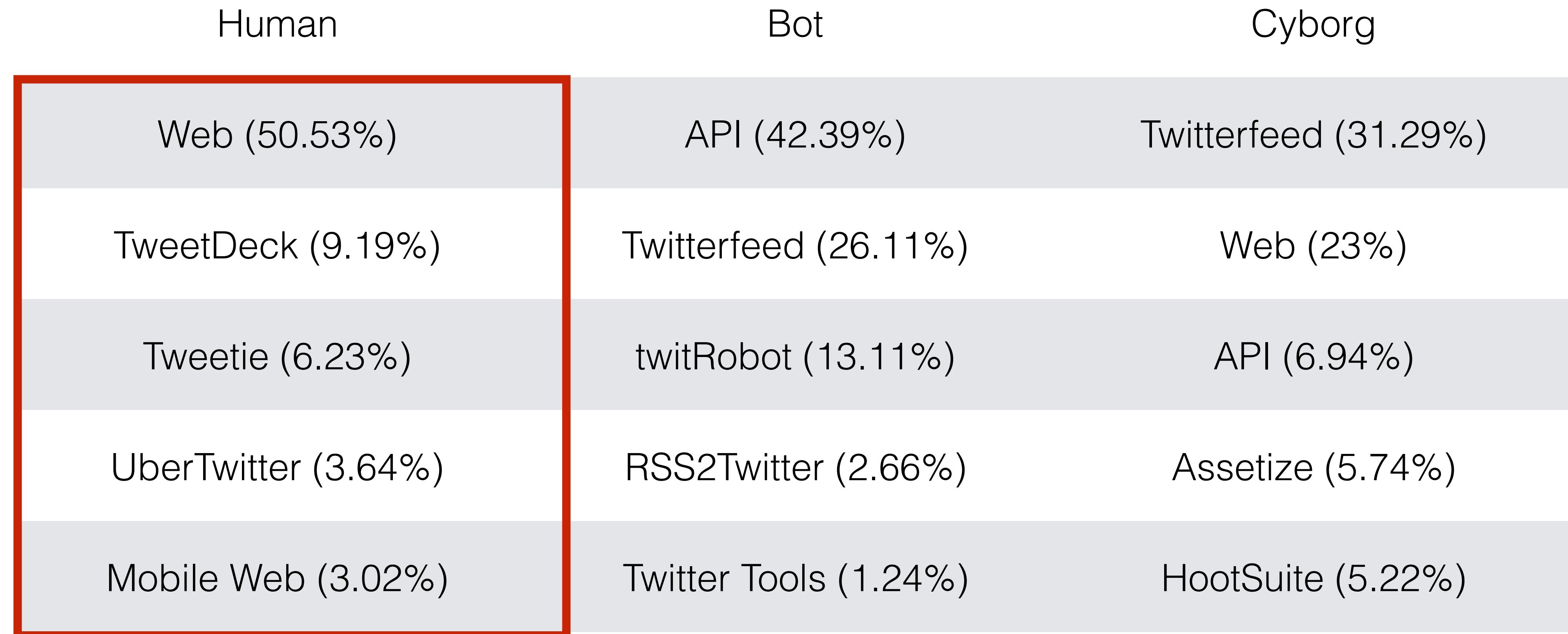
Fig. 6. Tweeting device makeup.



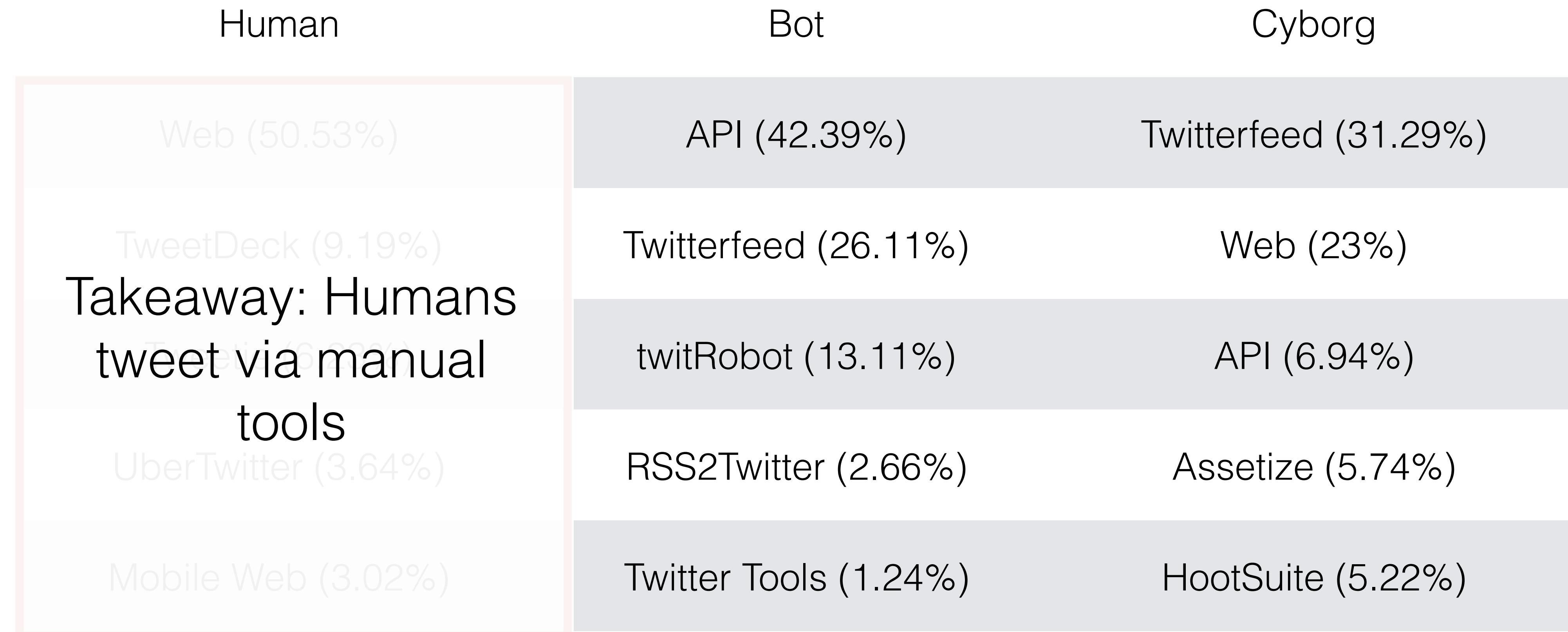
Tweeting Habits: What tools are used to Tweet?



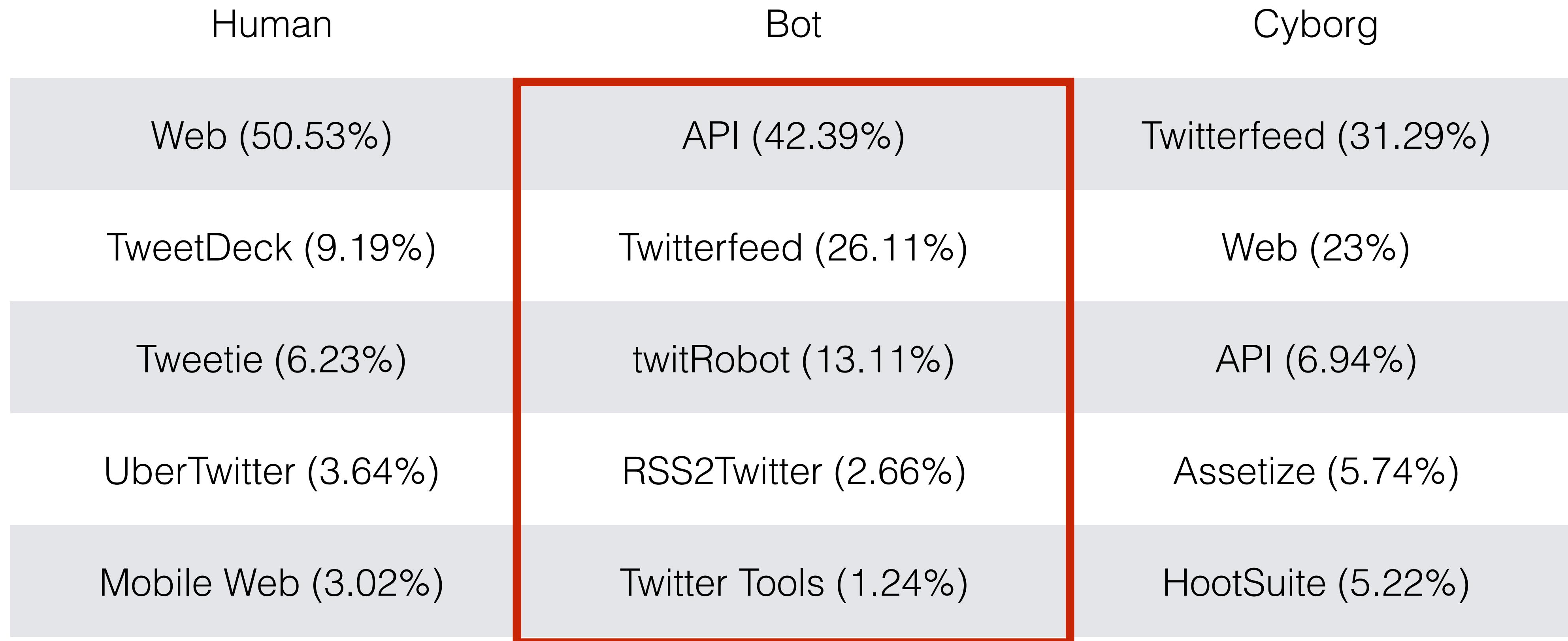
Tweeting Habits: What tools are used to Tweet?



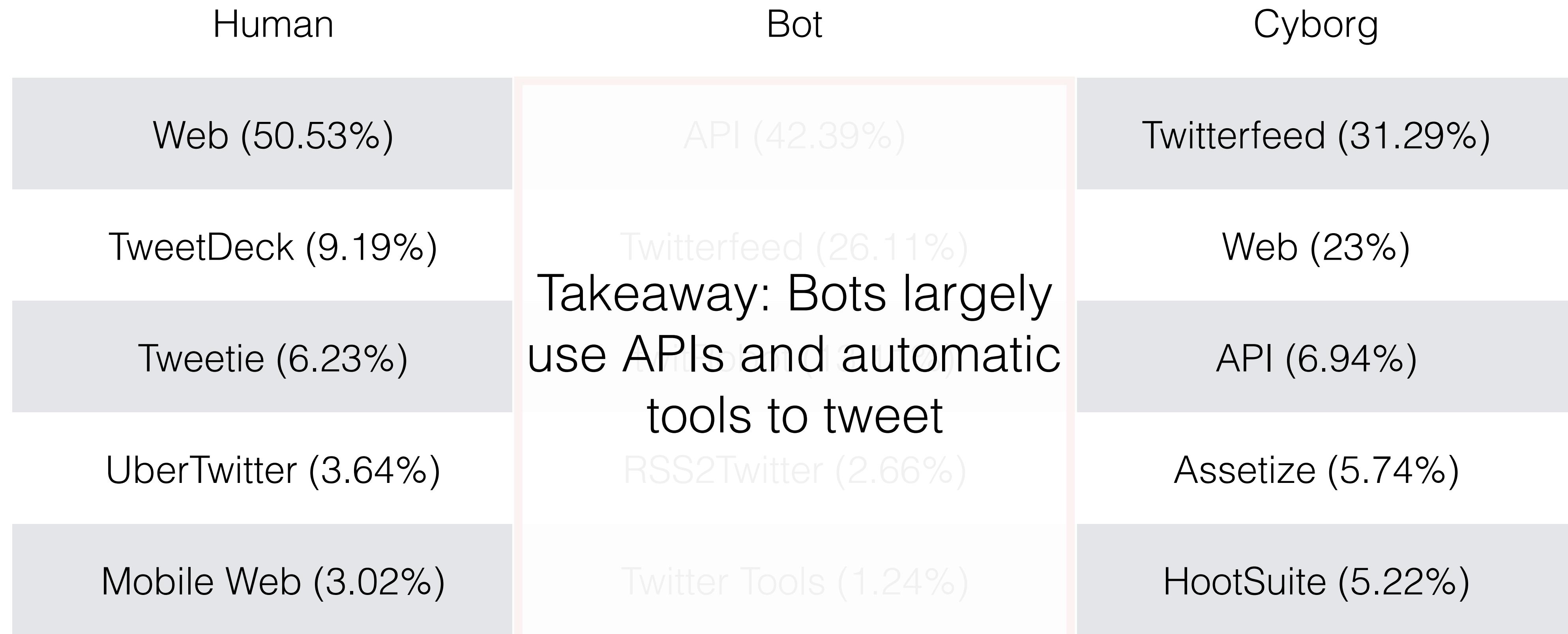
Tweeting Habits: What tools are used to Tweet?



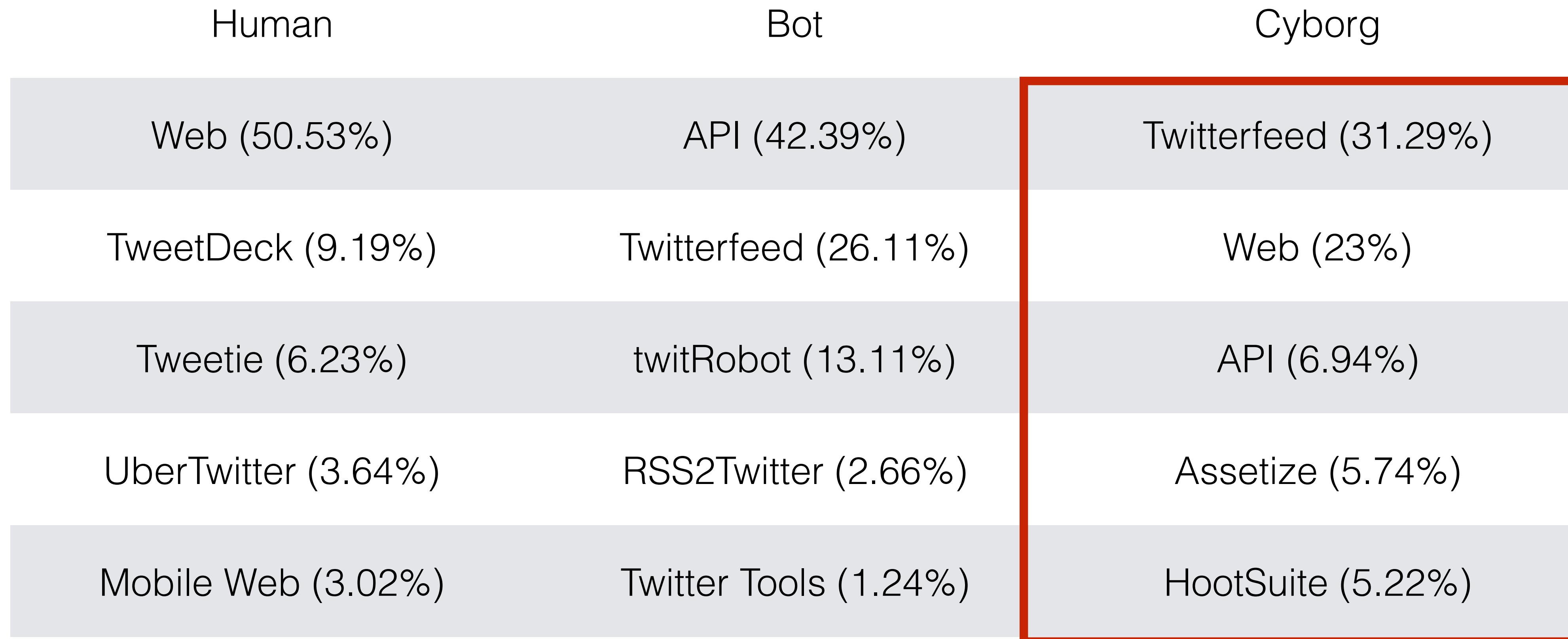
Tweeting Habits: What tools are used to Tweet?



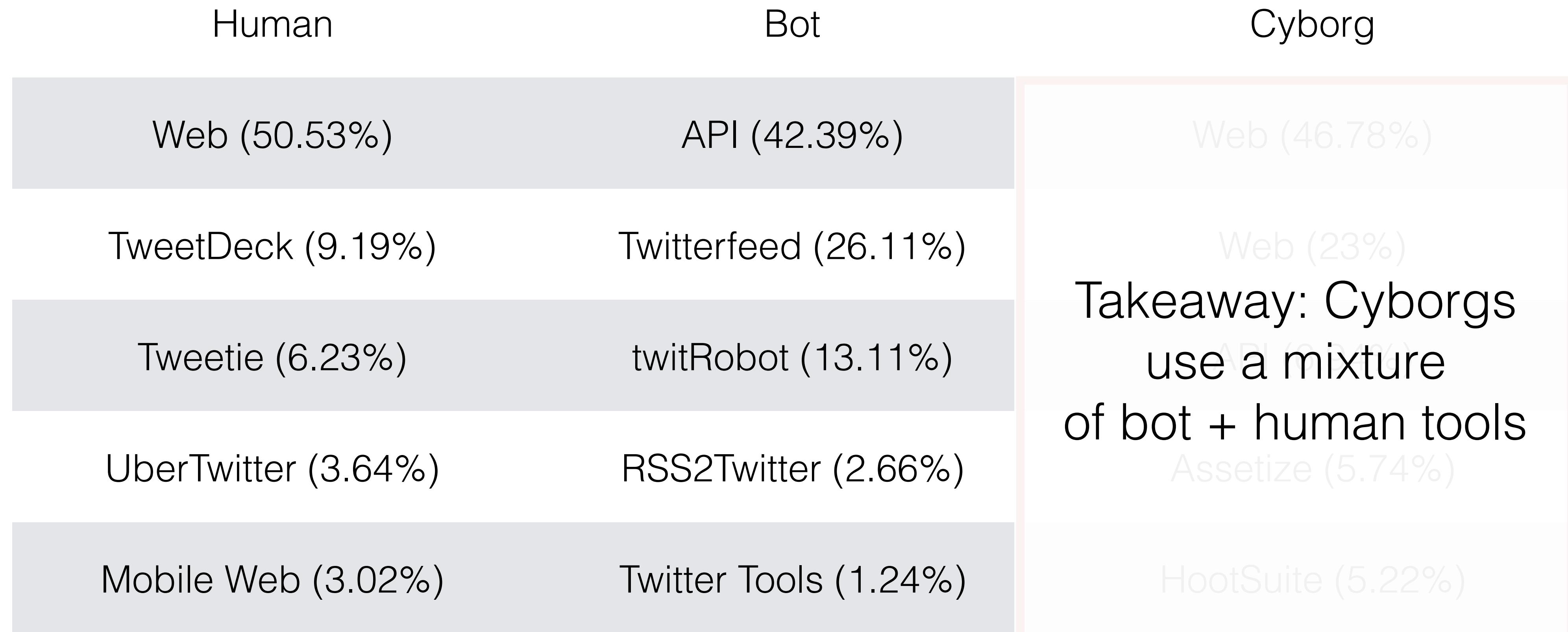
Tweeting Habits: What tools are used to Tweet?



Tweeting Habits: What tools are used to Tweet?



Tweeting Habits: What tools are used to Tweet?



Behavioral Patterns

Social Relationships

Tweeting Habits

Tweet Properties



Tweet Properties: URLs

- The # of URLs in bot tweets is higher than those of human + cyborg
- Some bots tweet more than one URL per tweet!

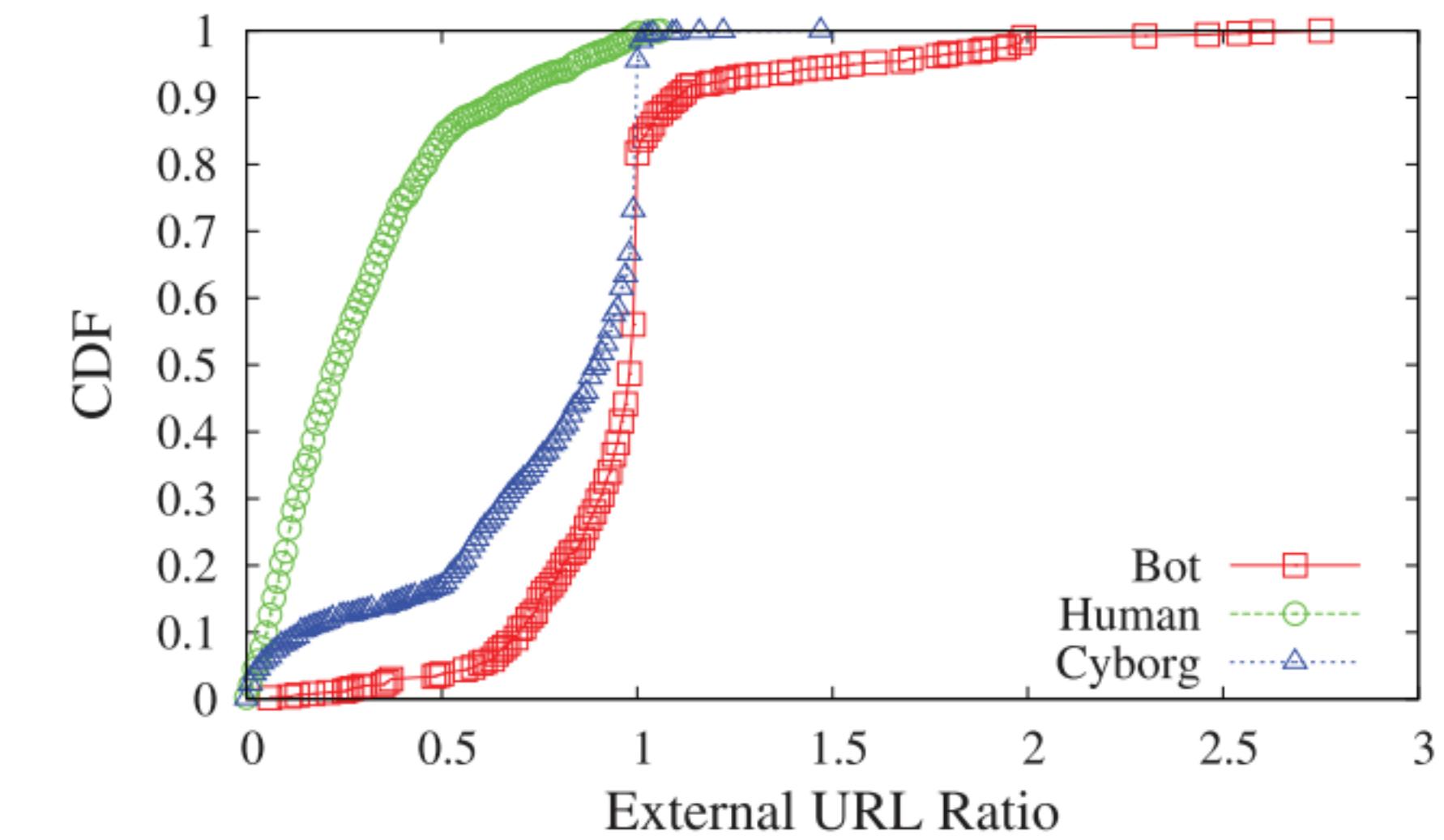


Fig. 7. External URL ratio in tweets.



Tweet Properties: URLs

- The # of URLs in bot tweets is higher than those of human + cyborg
- Some bots tweet more than one URL per tweet!
- Another useful property of URLs: appearing in **blacklists**

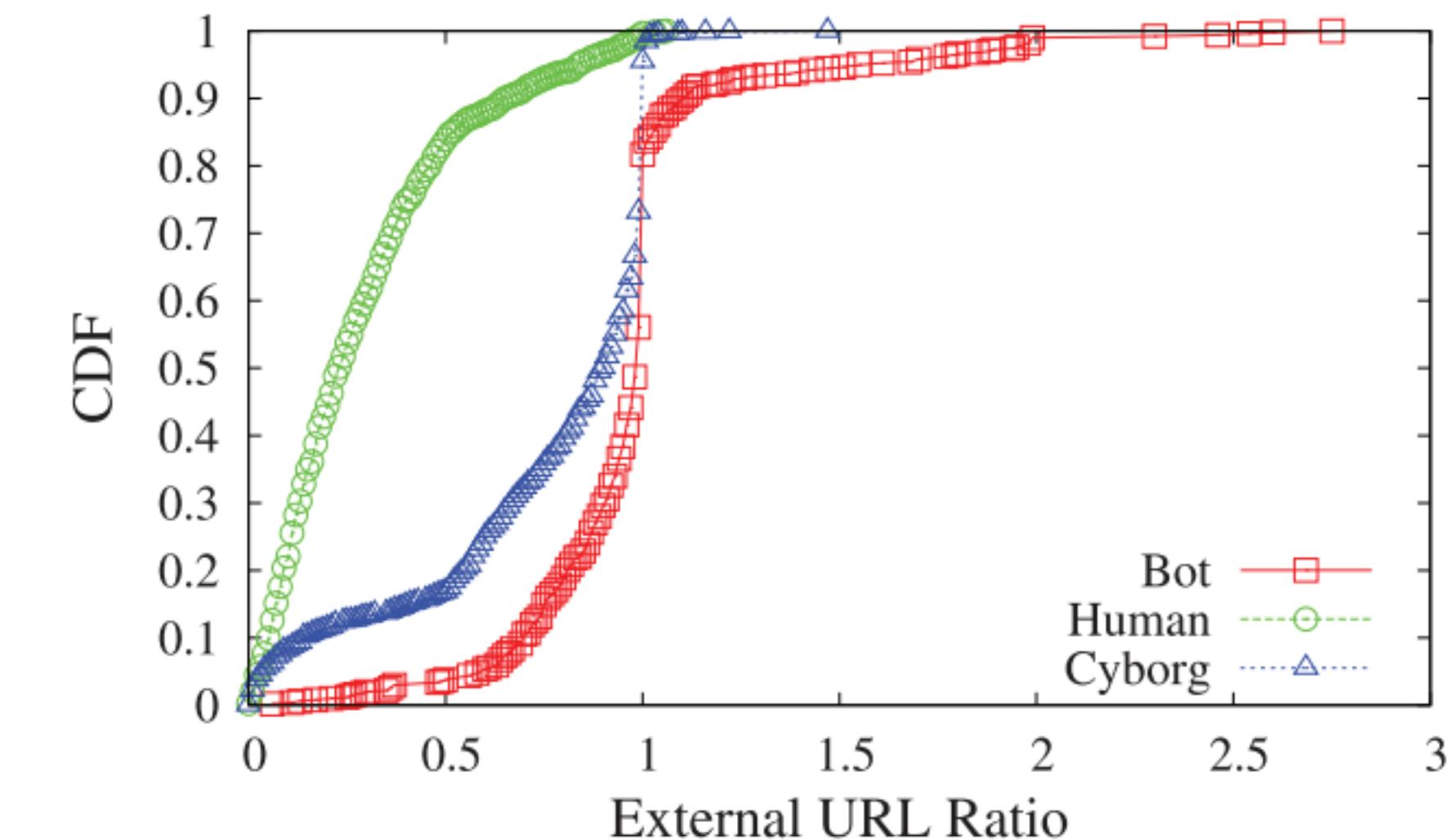


Fig. 7. External URL ratio in tweets.



Tweet Properties: Spam

- Intuition: Use **spam** as a mechanism for identifying automation



Tweet Properties: Spam

- Intuition: Use **spam** as a mechanism for identifying automation
- How? Naive Bayes for Spam Classification



Identifying Spam via Naive Bayes

- Simple classification model built off of conditional probabilities
 - “What is the likelihood a message is spam given previous spam messages?”
 - Treat each word (or set of words!) as *independent events*
 - Researchers labeled tweets as “spam” in ground-truth collection

$$\Pr(S|W) = \frac{\Pr(W|S) \cdot \Pr(S)}{\Pr(W|S) \cdot \Pr(S) + \Pr(W|H) \cdot \Pr(H)}$$



**Takeaway: Humans, Bots,
and Cyborgs exhibit
different behavioral patterns**



Can a standard machine
learning classifier use these
differences to detect bots?



Classifying Automation: Features

Features

Timing Entropy

URL Ratio

Automated Device %

Bayesian Spam Detection

Manual Device %

Registration Date

Mention Ratio

Link Safety

Hashtag Ratio

Followers to Friends Ratio

Account Verification



Classifying Automation: Features

Features

Timing Entropy

URL Ratio

Automated Device %

Bayesian Spam Detection

Manual Device %

Registration Date

Mention Ratio

Link Safety

Hashtag Ratio

Followers to Friends Ratio

Account Verification



Classifying Automation: Training and Testing

- Use a *random forest*, popular ML algorithm for labeled classification
- Use 10-fold cross validation to determine accuracy of the model
- Split ground-truth data randomly into 10 partitions
 - Train on 9, test on 1
 - Repeat and average results



Classifying Automation: Training and Testing

		Classified			
		Human	Cyborg	Bot	True Positive
Actual	Human	1972	27	1	98.6%
	Cyborg	65	1833	102	91.7%
	Bot	2	46	1952	97.6%
Total					96.0%



Classifying Automation: Training and Testing

		Classified			
		Human	Cyborg	Bot	True Positive
Actual	Human	1972	27	1	98.6%
	Cyborg	65	1833	102	91.7%
	Bot	2	46	1952	97.6%
Total					96.0%



Classifying Automation: Training and Testing

		Classified			
		Human	Cyborg	Bot	True Positive
Actual	Human	1972	27	1	98.6%
	Cyborg	65	1833	102	91.7%
	Bot	2	46	1952	97.6%
Total					96.0%



Twitter Composition

- Apply classification algorithm to set of 500,000 users
- 53.2% Human
- 36.2% Cyborg
- 10.5% Bot
- 5:4:1 Ratio on Twitter in 2012



My Takeaways

- Good first step towards automation detection (96% TP!), but shelf life of technique is limited
 - Behavioral properties are the easiest to change
 - In 2018, bots behave just like humans



My Takeaways

- Good first step towards automation detection (96% TP!), but shelf life of technique is limited
 - Behavioral properties are the easiest to change
 - In 2018, bots behave just like humans
- We still don't understand how effective bots are at their tasks



My Takeaways

- Good first step towards automation detection (96% TP!), but shelf life of technique is limited
 - Behavioral properties are the easiest to change
 - In 2018, bots behave just like humans
- We still don't understand how effective bots are at their tasks
- **Paper: Understanding the network infrastructure behind automation**
- **Paper: Attribution of bot effectiveness through network analysis**



Classifying Users on Twitter: Ground Truth

	Human	Bot	Cyborg
Criteria	<ul style="list-style-type: none">• Original, intelligent, specific content	<ul style="list-style-type: none">• Unoriginal content• <i>Clearly</i> automated (i.e., RSS feed)• Spammy, filled with shady URLs	<ul style="list-style-type: none">• Evidence of both bot and human participation



Classifying Automation: Feature Importance

Features	Accuracy
Timing Entropy	82.8%
URL Ratio	74.9%
Automated Device %	71.0%
Bayesian Spam	69.5%
Manual Device %	69.2%
Registration Date	62.9%
Mention Ratio	56.2%
Link Safety	49.3%
Hashtag Ratio	47.0%
Followers to Friends	45.3%
Account Verification	35.0%



Classifying Automation: Feature Importance

Features Accuracy

Takeaway: No one
feature tells the whole

Manual Device %	69.2%
Registration Late	52.9%
Mention Ratio	56.2%
Link Safety	49.3%
Hashtag Ratio	47.0%
Followers to Friends	45.3%
Account Verification	35.0%



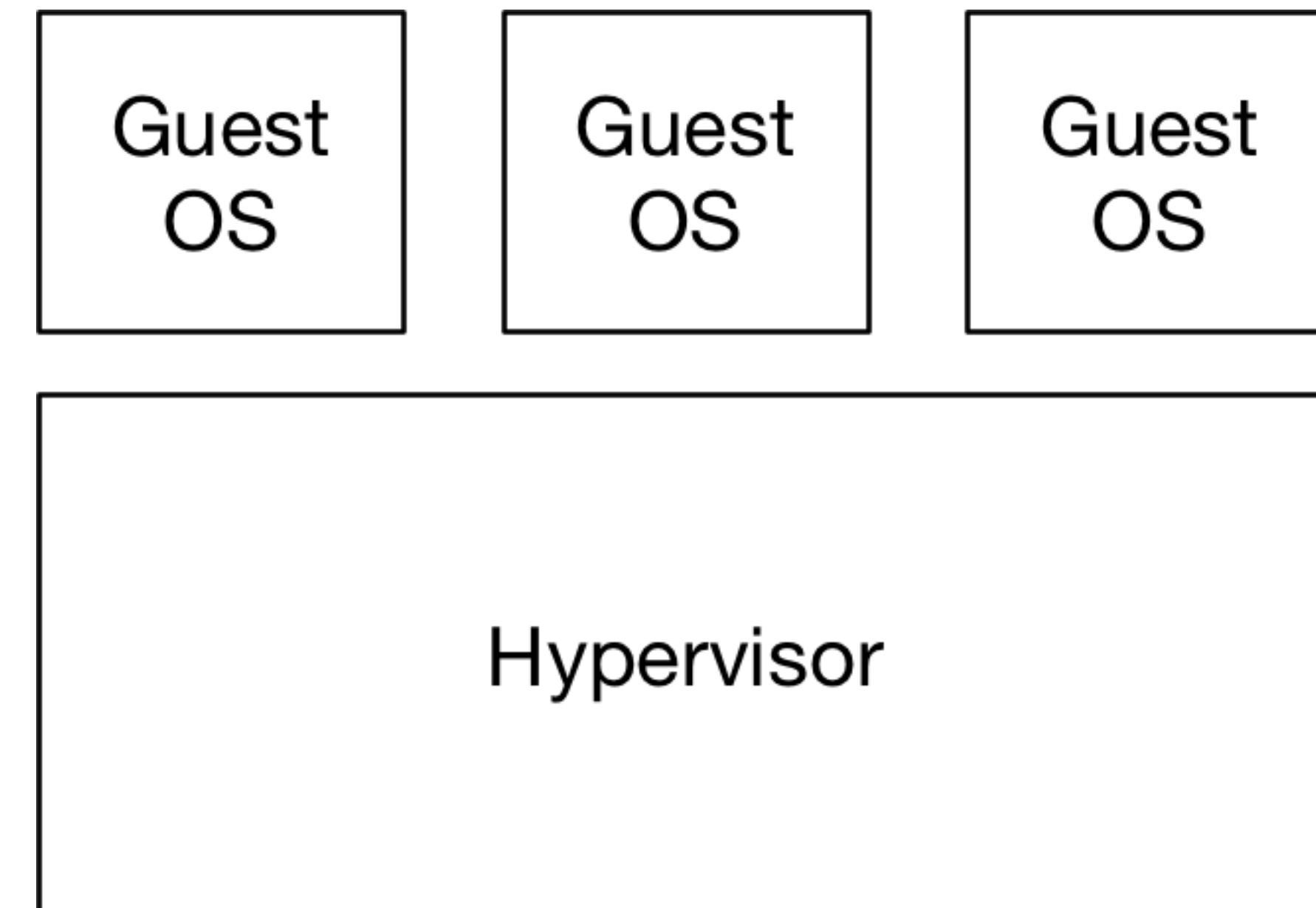
The Turtles Project: Design and Implementation of Nested Virtualization

Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, Ben-Ami Yassour



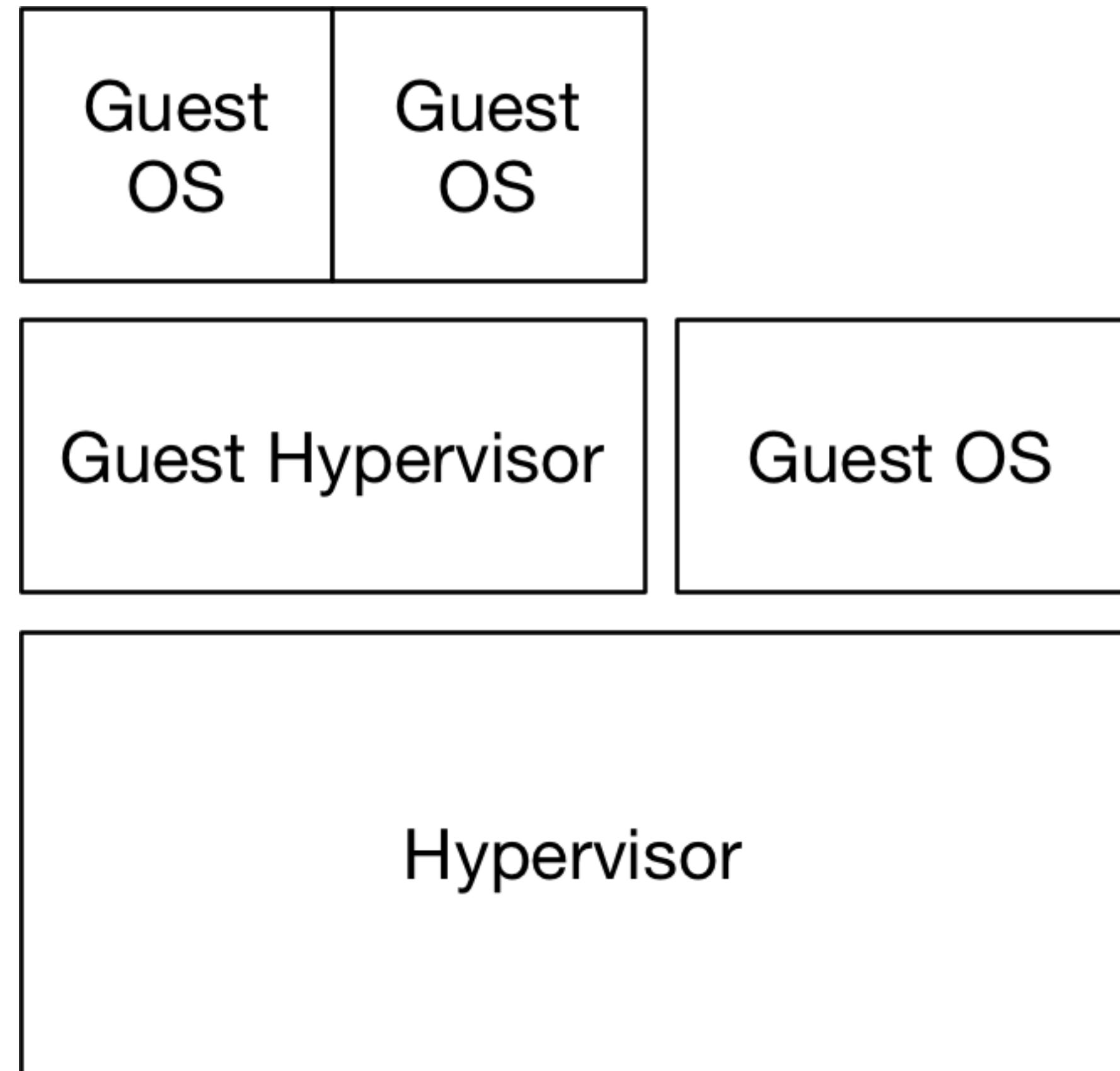
Classic Virtualization

- One hypervisor
 - Runs multiple guest operating systems
 - Hardware support from Intel (Intel VMX), AMD (AMD SVM) for x86



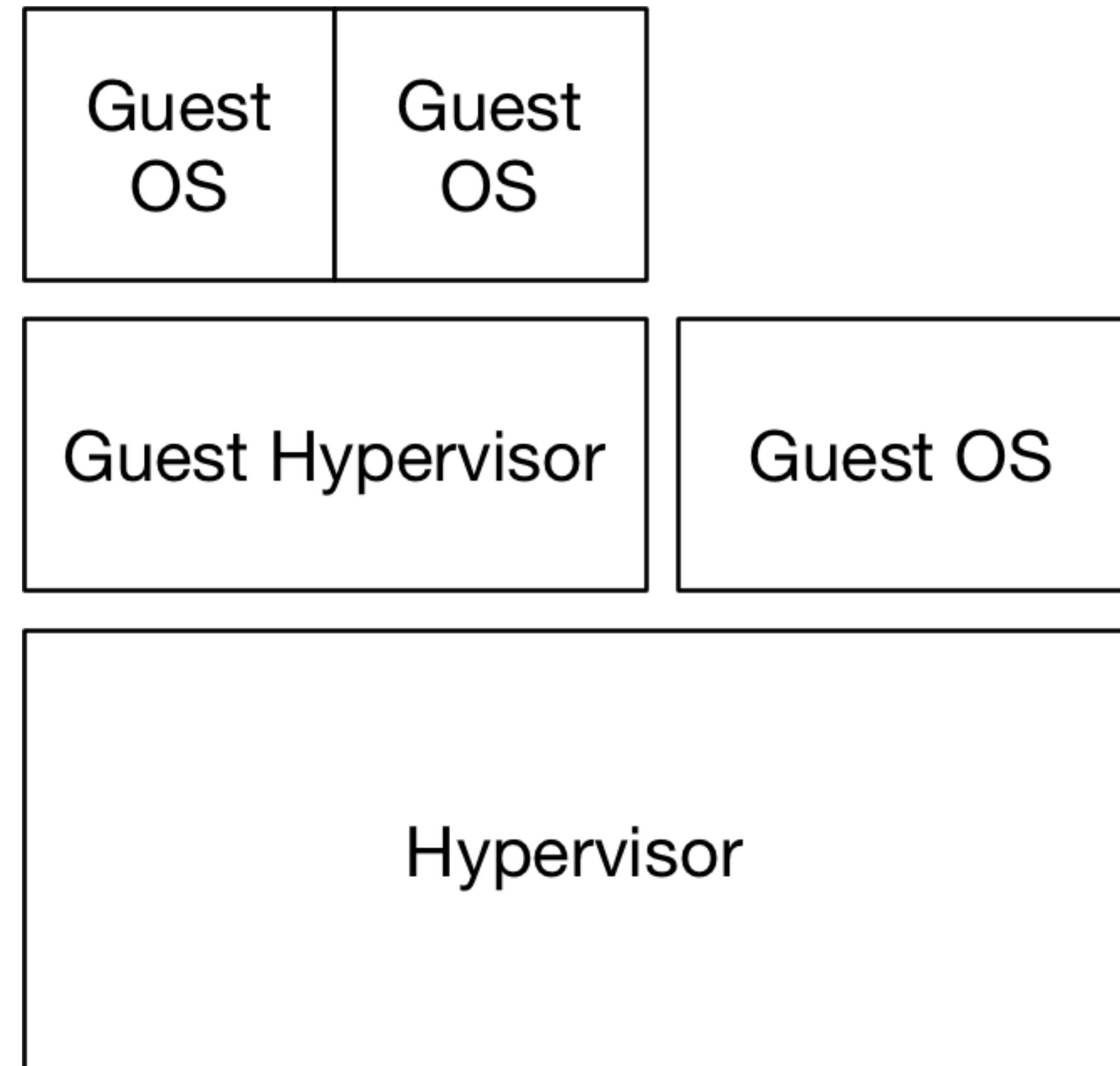
Nested Virtualization

- Run multiple *unmodified* hypervisors
 - Unmodified guest operating systems



Nested Virtualization

- Run multiple *unmodified* hypervisors
 - Unmodified guest operating systems
 - Nested virtualization increasing in importance
 - Many commodity OSes make use of virtualization already



Why Nested Virtualization?

- Cloud
 - Expose user-controlled hypervisor as a virtual machine



Why Nested Virtualization?

- Cloud
 - Expose user-controlled hypervisor as a virtual machine
- Security
 - Hypervisor-level rootkit protection, hypervisor-level intrusion detection



Why Nested Virtualization?

- Cloud
 - Expose user-controlled hypervisor as a virtual machine

Problem: No x86 support for nested virtualization!

- Security
 - Hypervisor-level rootkit protection, hypervisor-level intrusion detection



Why Nested Virtualization?

- Cloud
 - Expose user machine
 - Scale infrastructure on host
- Security
 - Hypervisor-level detection



The Turtles Project

- *Single-level architectural support for nested virtualization*



The Turtles Project

- *Single-level architectural support for nested virtualization*
 - Single hypervisor mode



The Turtles Project

- *Single-level architectural support for nested virtualization*
 - Single hypervisor mode
 - Any trap at a nested level is handled by the “base” hypervisor, called L_0



The Turtles Project

- *Single-level architectural support for nested virtualization*
 - Single hypervisor mode
 - Any trap at a nested level is handled by the “base” hypervisor, called L_0
 - Traps are then *forwarded* by L_0 to the appropriate level

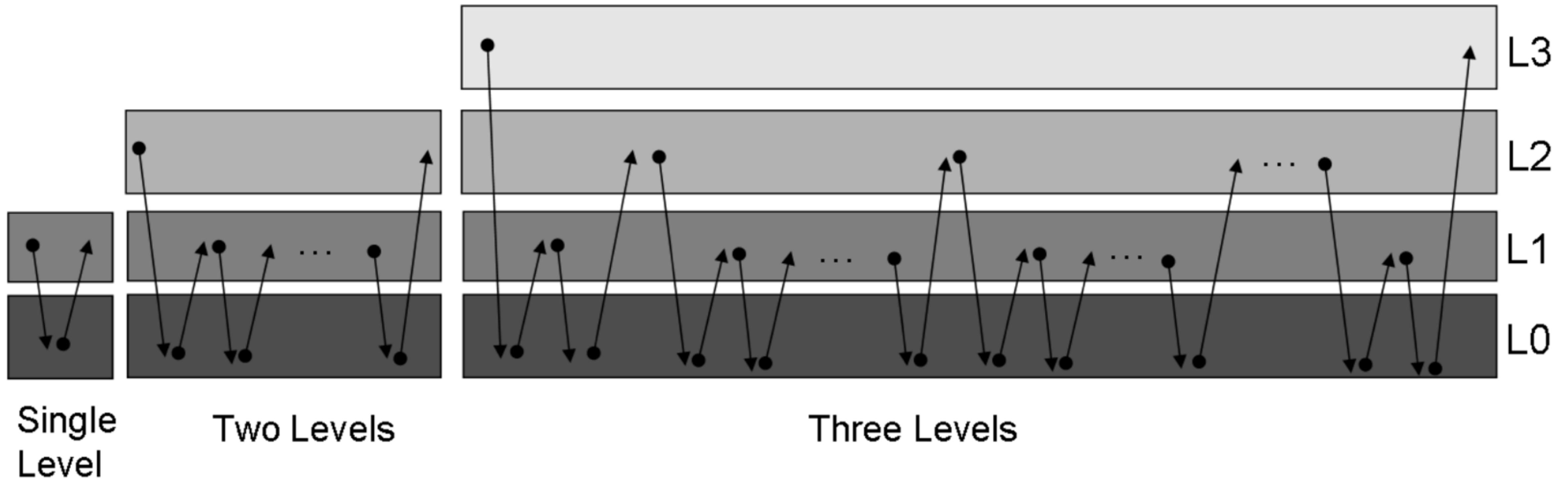


The Turtles Project

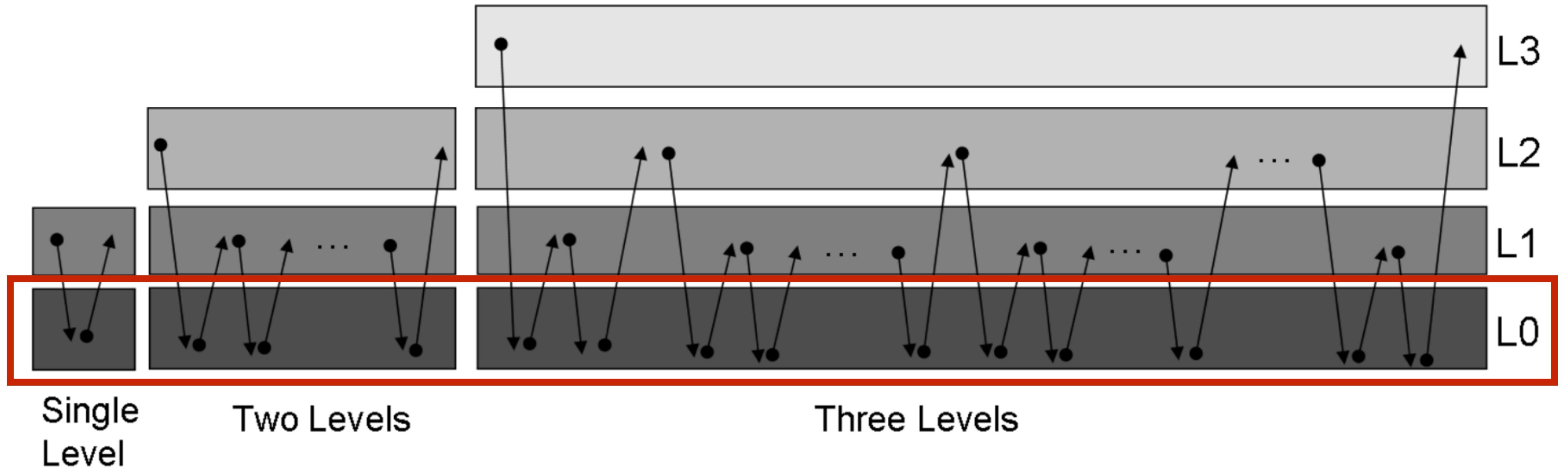
- *Single-level architectural support for nested virtualization*
 - Single hypervisor mode
 - Any trap at a nested level is handled by the “base” hypervisor, called L_0
 - Traps are then *forwarded* by L_0 to the appropriate level
 - L_0 thus *multiplexes* hardware between higher level hypervisors and OSes



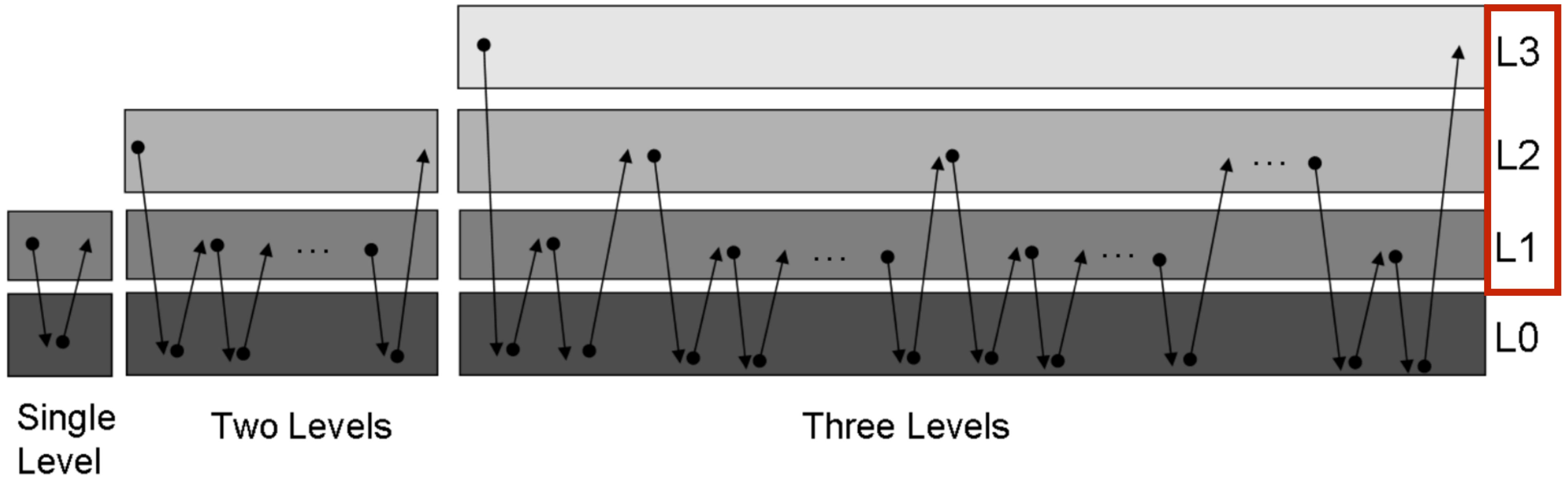
Turtles Theory



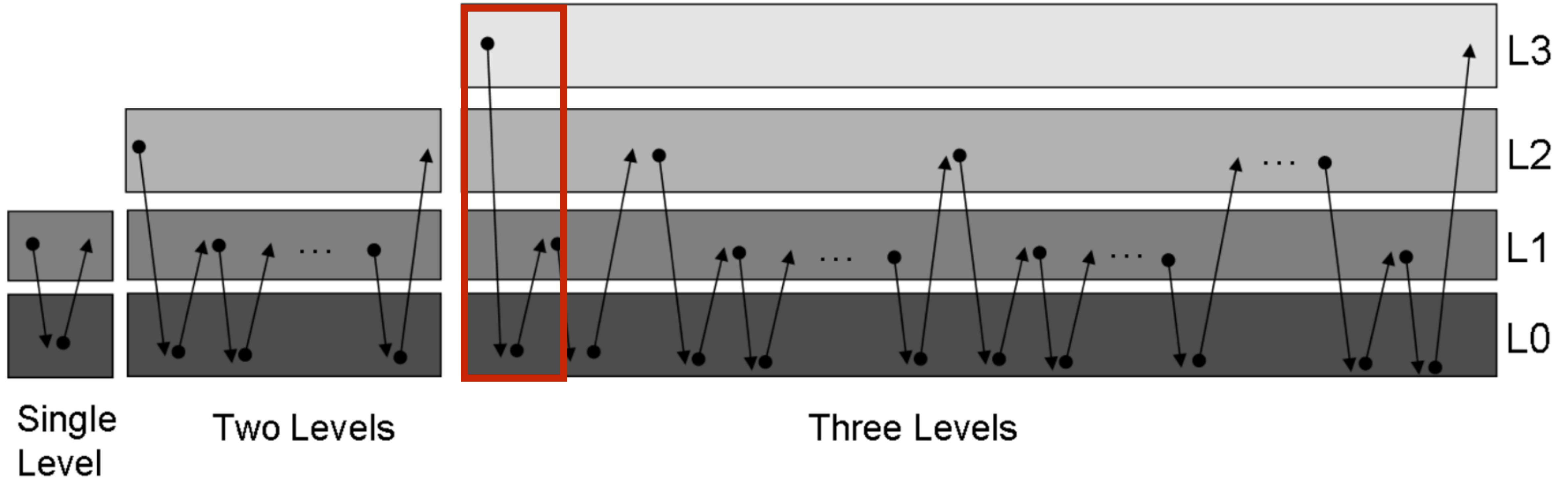
Turtles Theory



Turtles Theory



Turtles Theory



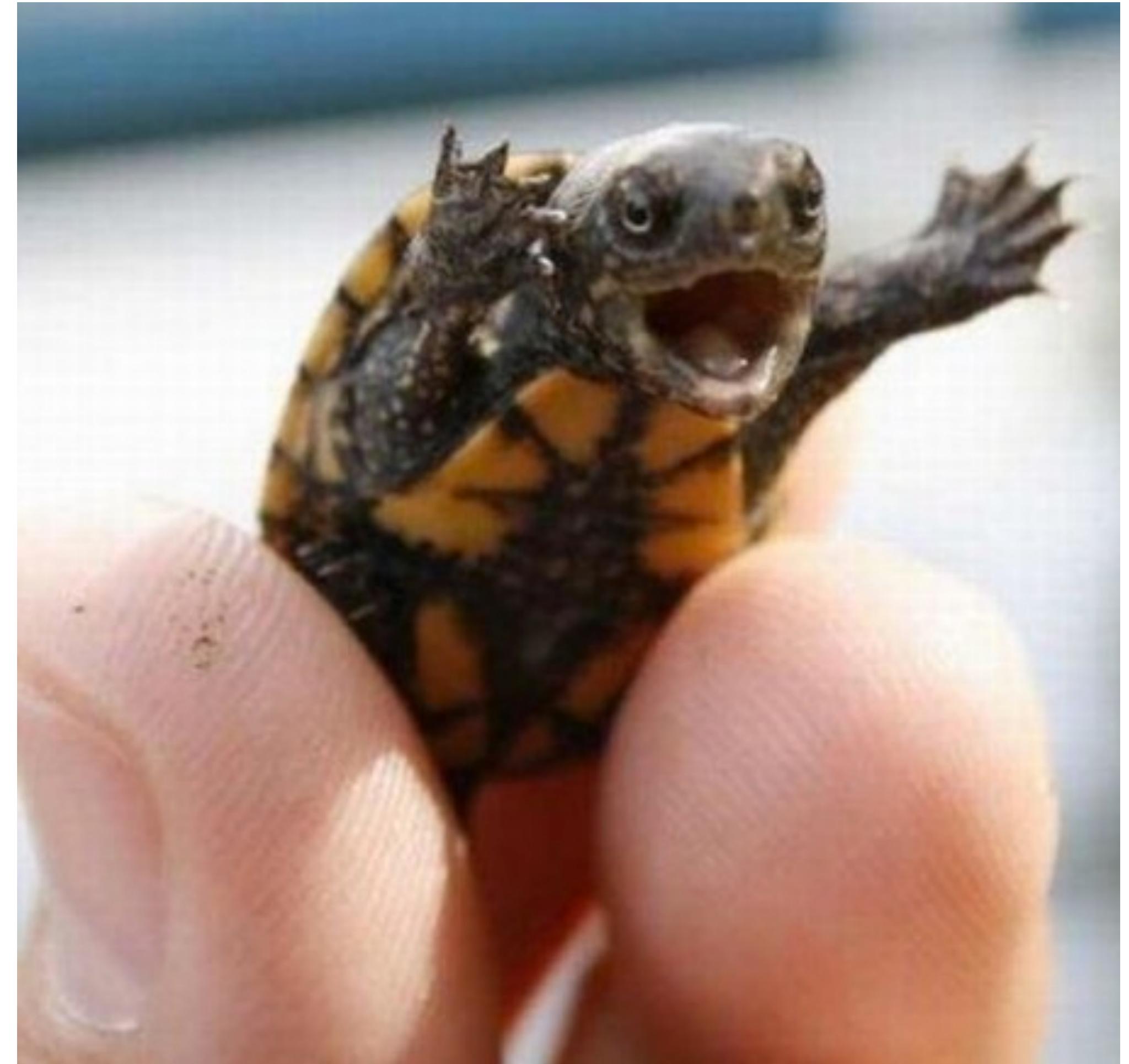
Implementing Turtles

- Nested VMX Virtualization for CPUs
- Multidimensional Paging for MMUs
- Multi-level Device Assignment for I/O



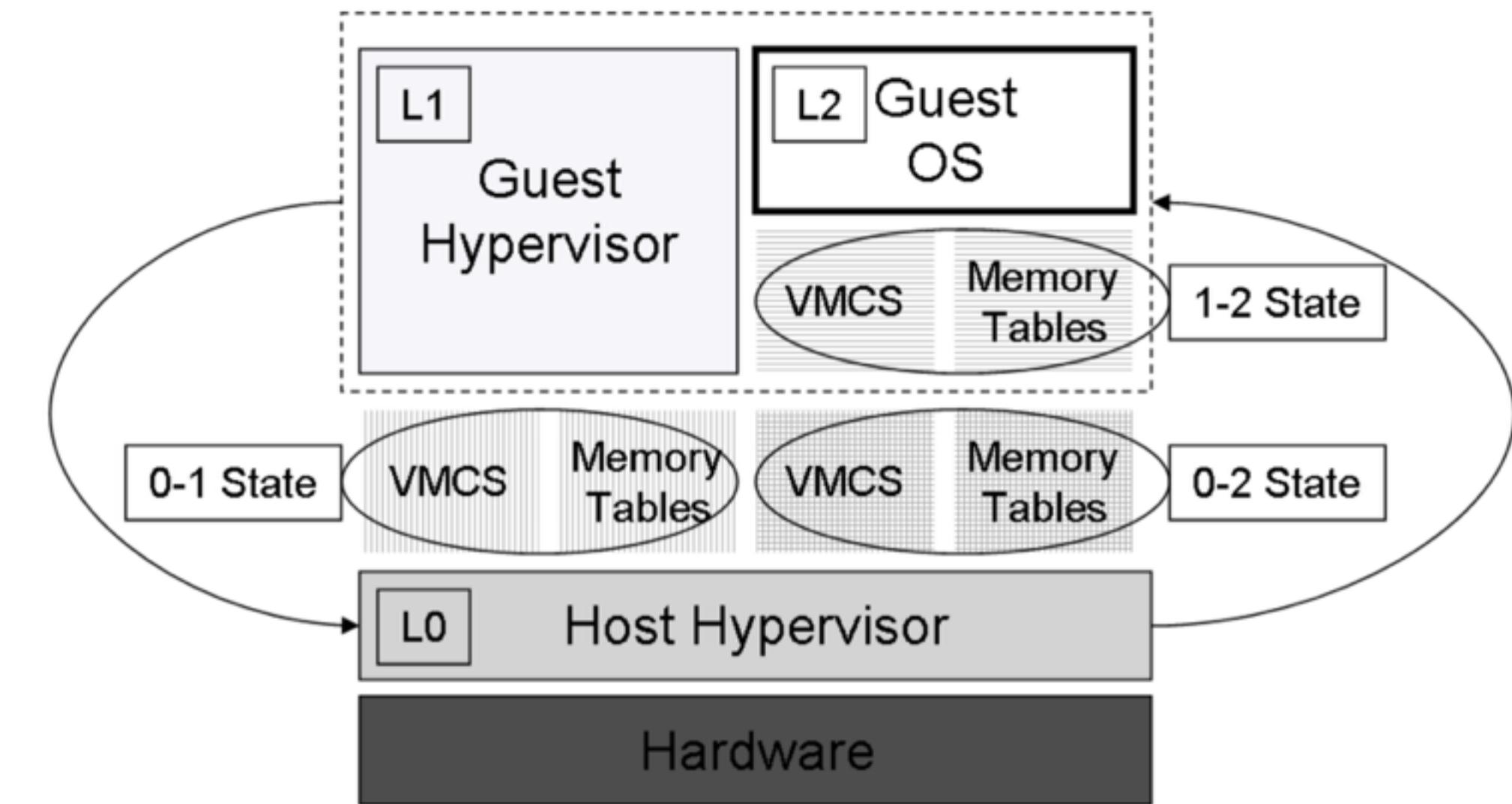
Implementing Turtles

- **Nested VMX Virtualization for CPUs**
- Multidimensional Paging for MMUs
- Multi-level Device Assignment for I/O



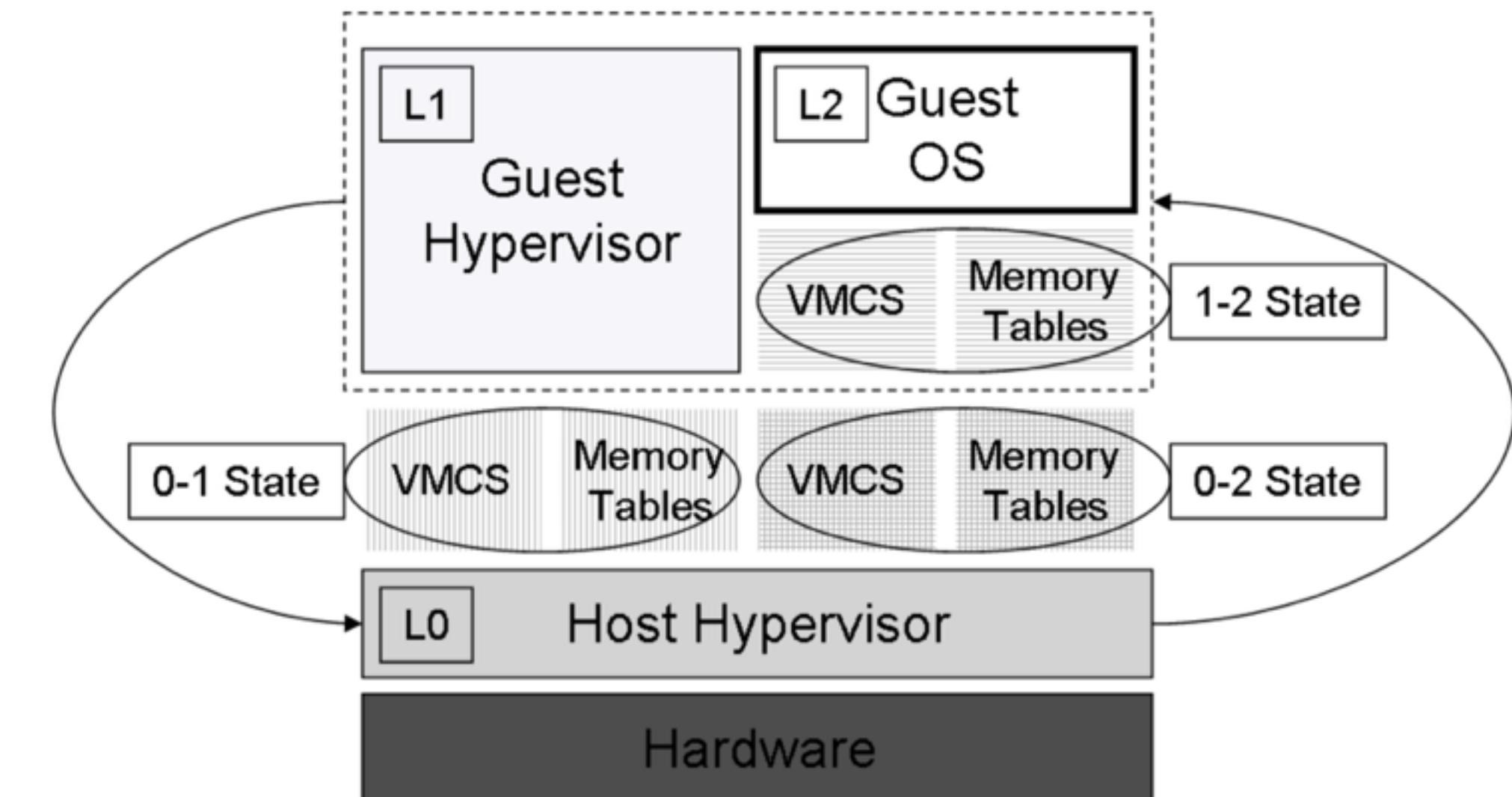
Nested CPU Virtualization: Design

- L₀ can leverage virtualization support from hardware (VMX)
 - Supports *guest mode* and *host mode* for CPUs



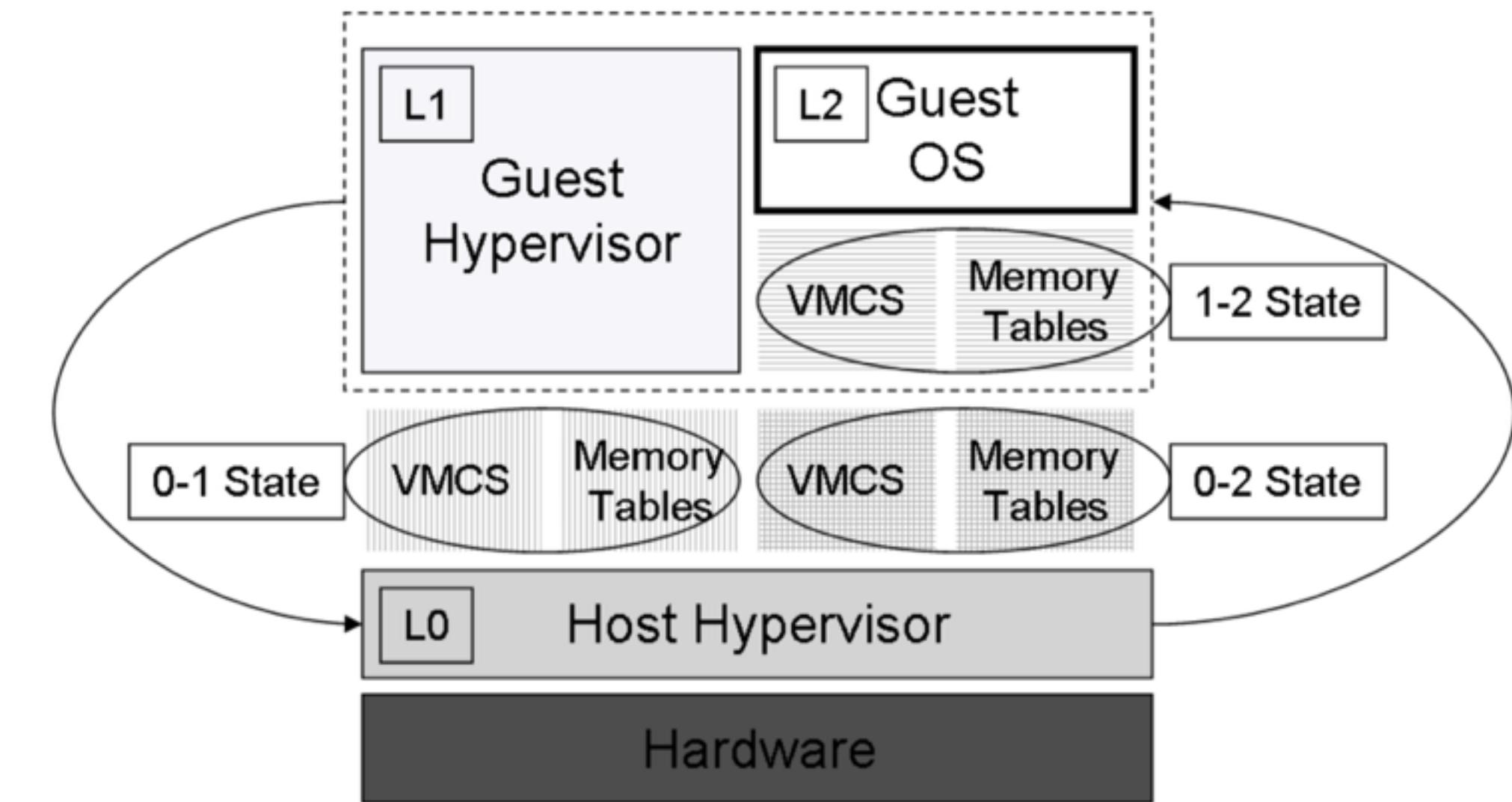
Nested CPU Virtualization: Design

- L₀ can leverage virtualization support from hardware (VMX)
 - Supports *guest mode* and *host mode* for CPUs
 - VMCS: Control structures for defining virtual machines environments
 - Defines traps, CPU registers, host state, guest state



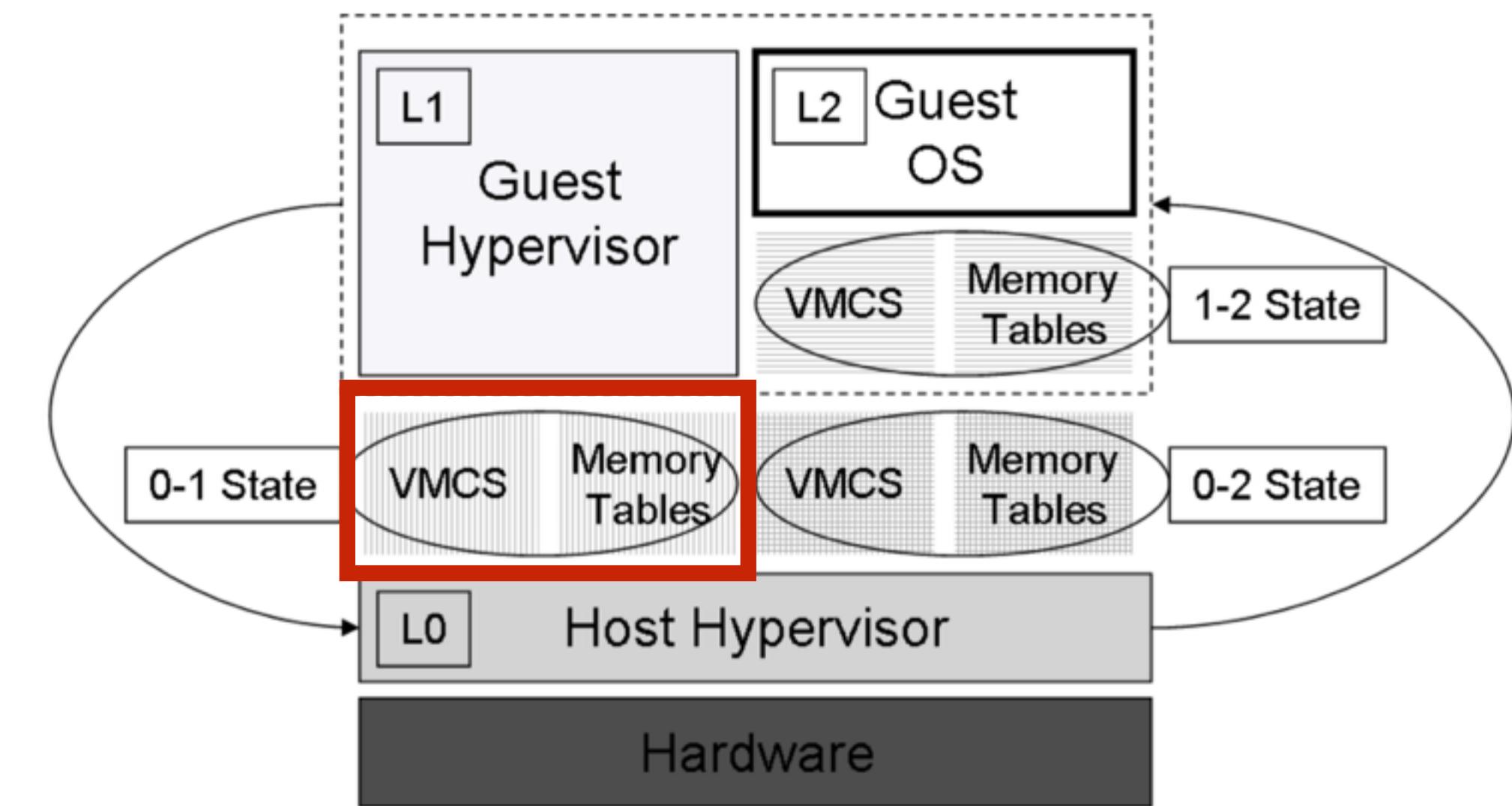
Nested CPU Virtualization: Design

- Recursive VMX emulation



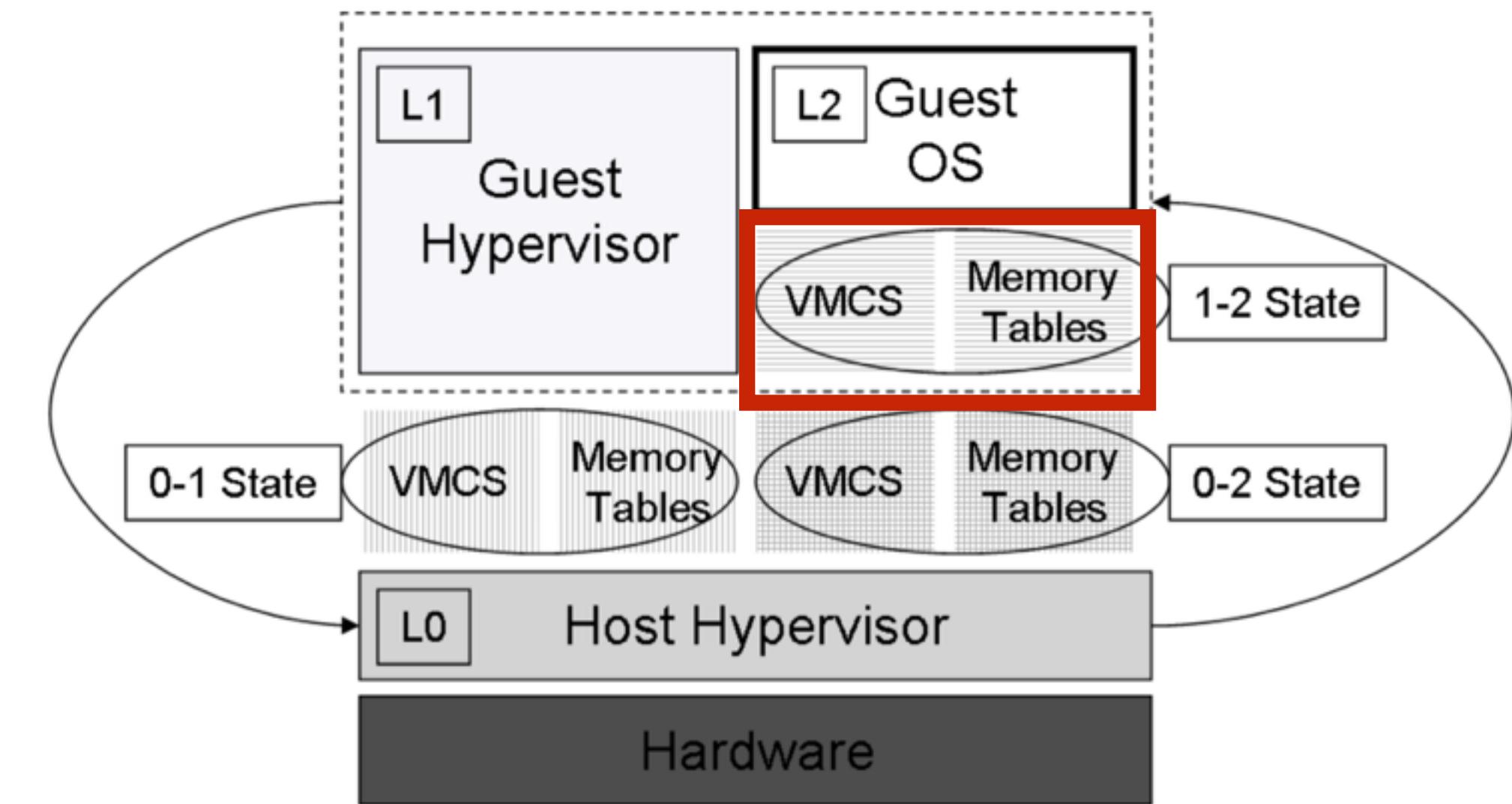
Nested CPU Virtualization: Design

- Recursive VMX emulation
 - L_0 needs to *emulate* VMX to L_1



Nested CPU Virtualization: Design

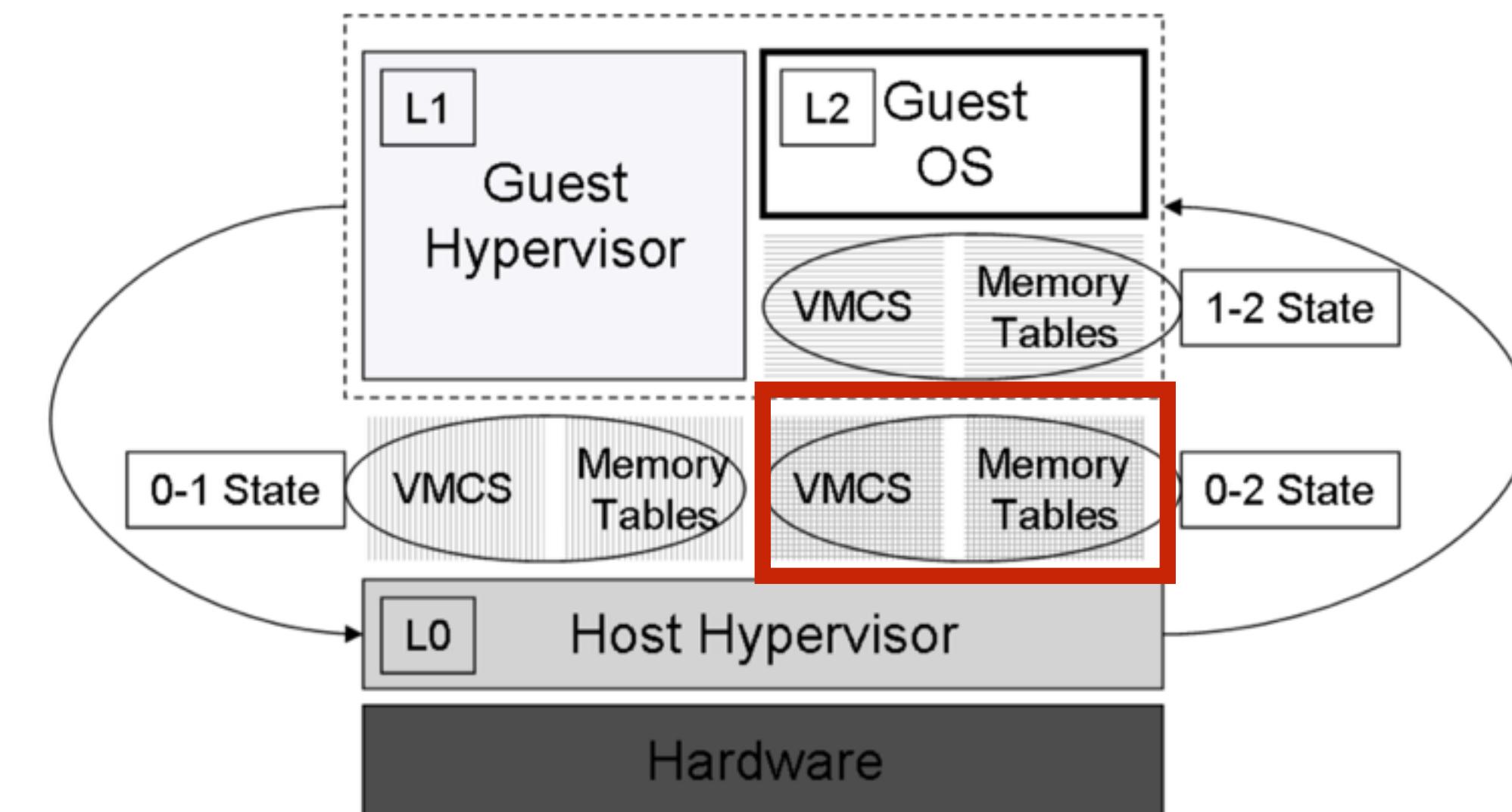
- Recursive VMX emulation
 - L_0 needs to *emulate* VMX to L_1
 - L_n will do the same for L_{n+1}



Nested CPU Virtualization: Design

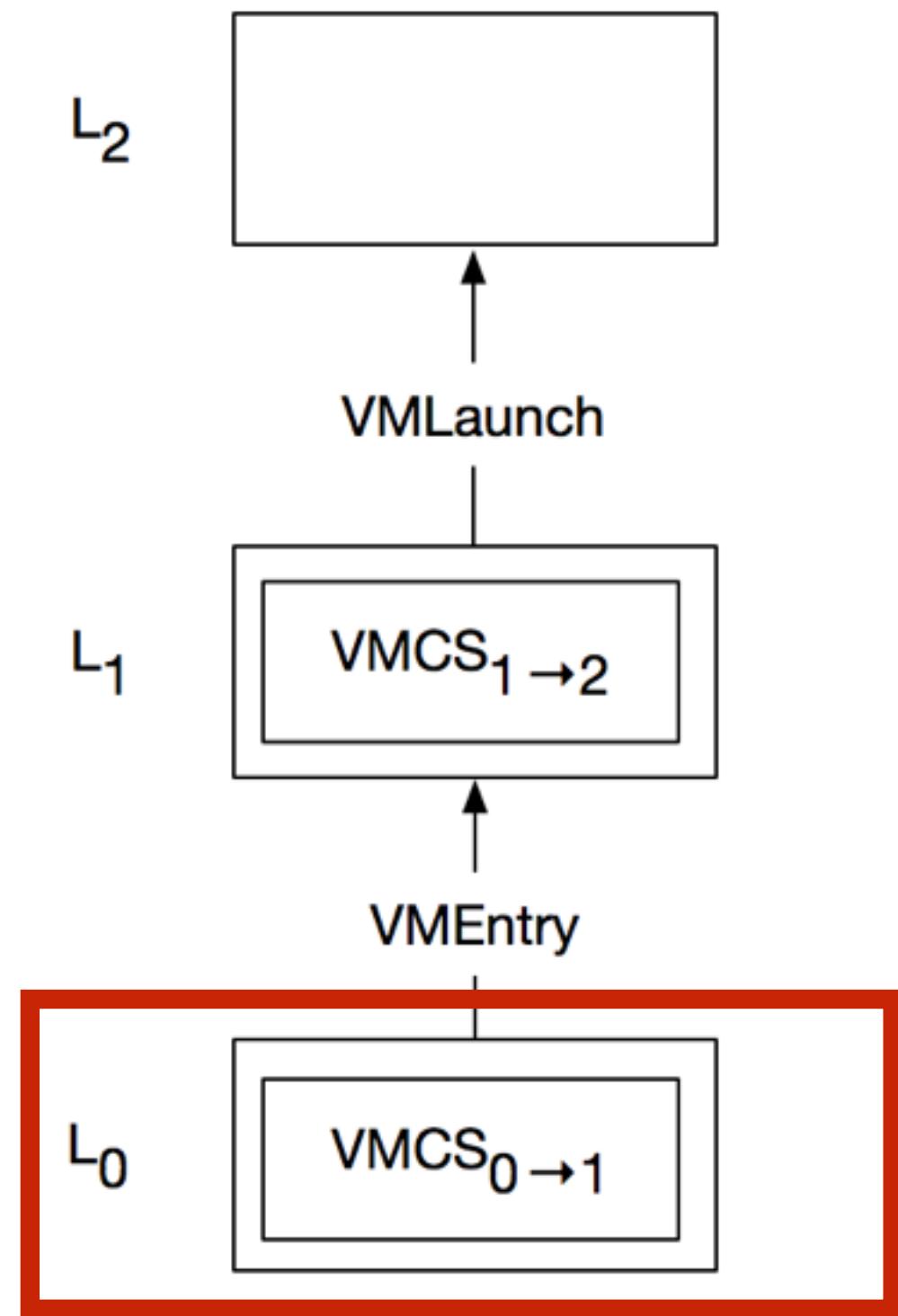
Key point: L₀ must maintain state (VMCS) about L₂ to actually run L₂ on the hardware

- run needs to map L₁ VMX to L₁
- L_n will do the same for L_{n+1}



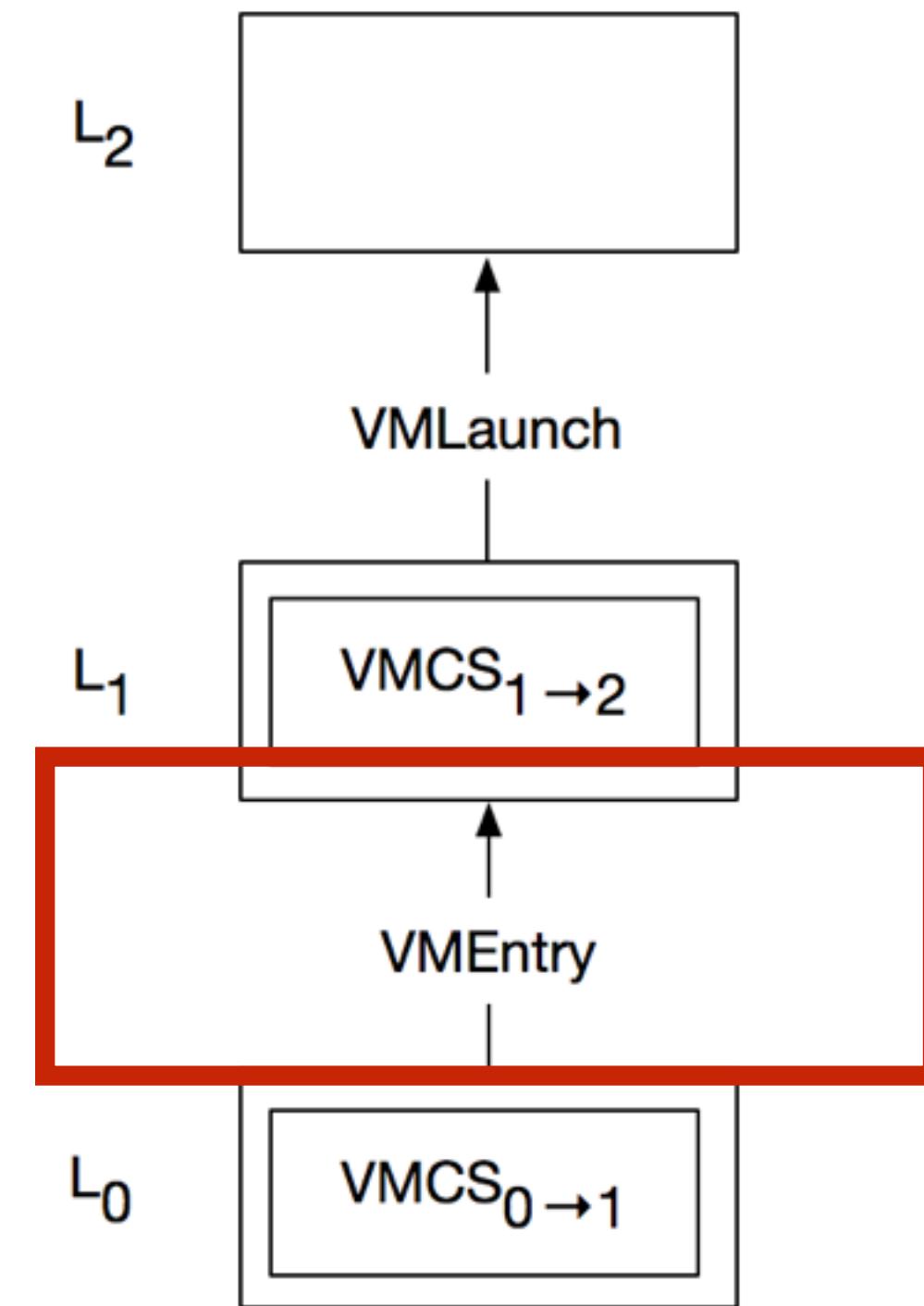
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$



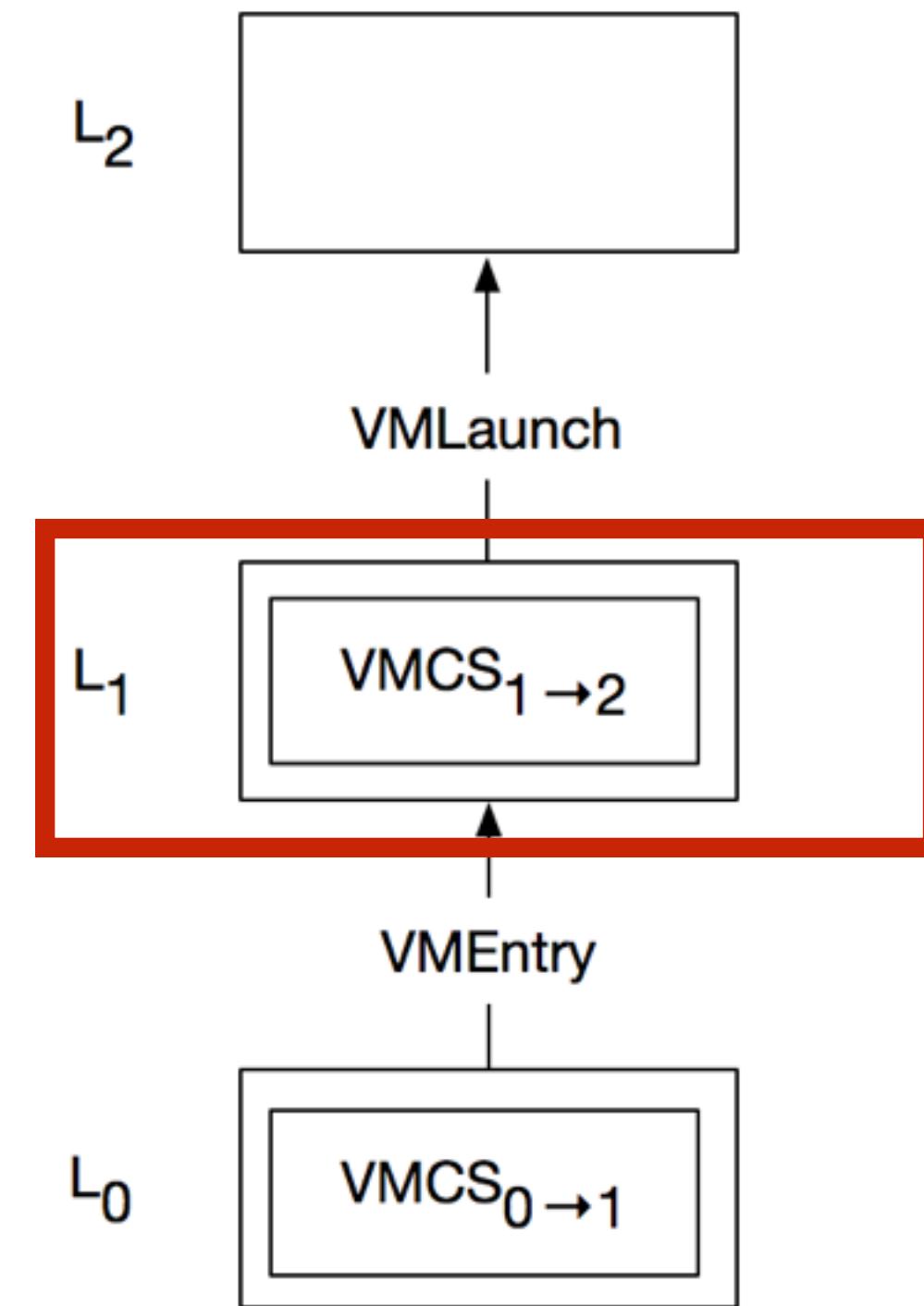
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$



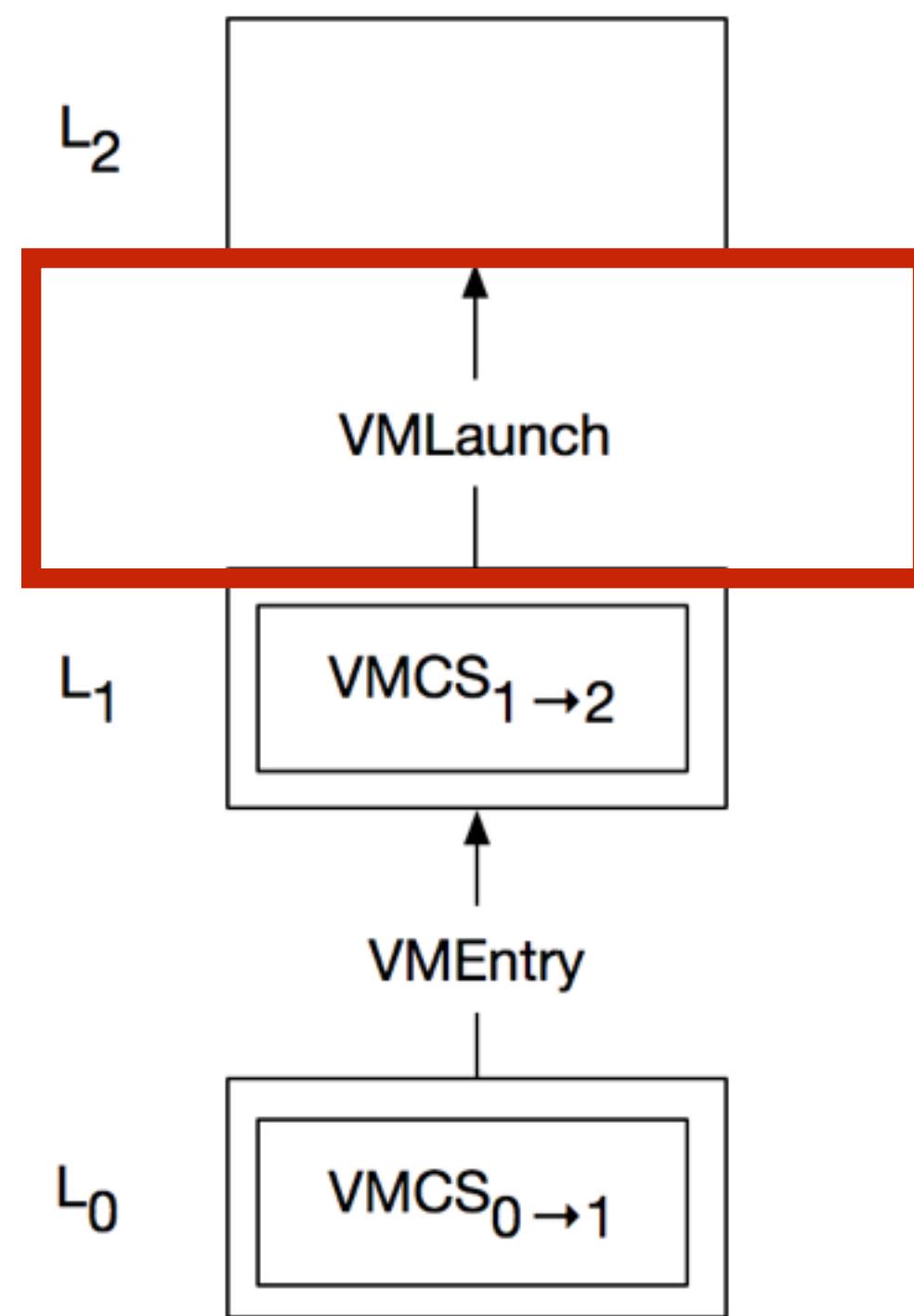
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$
- L_1 runs L_2 by creating $\text{VMCS}_{1 \rightarrow 2}$, then calls `vmlaunch`



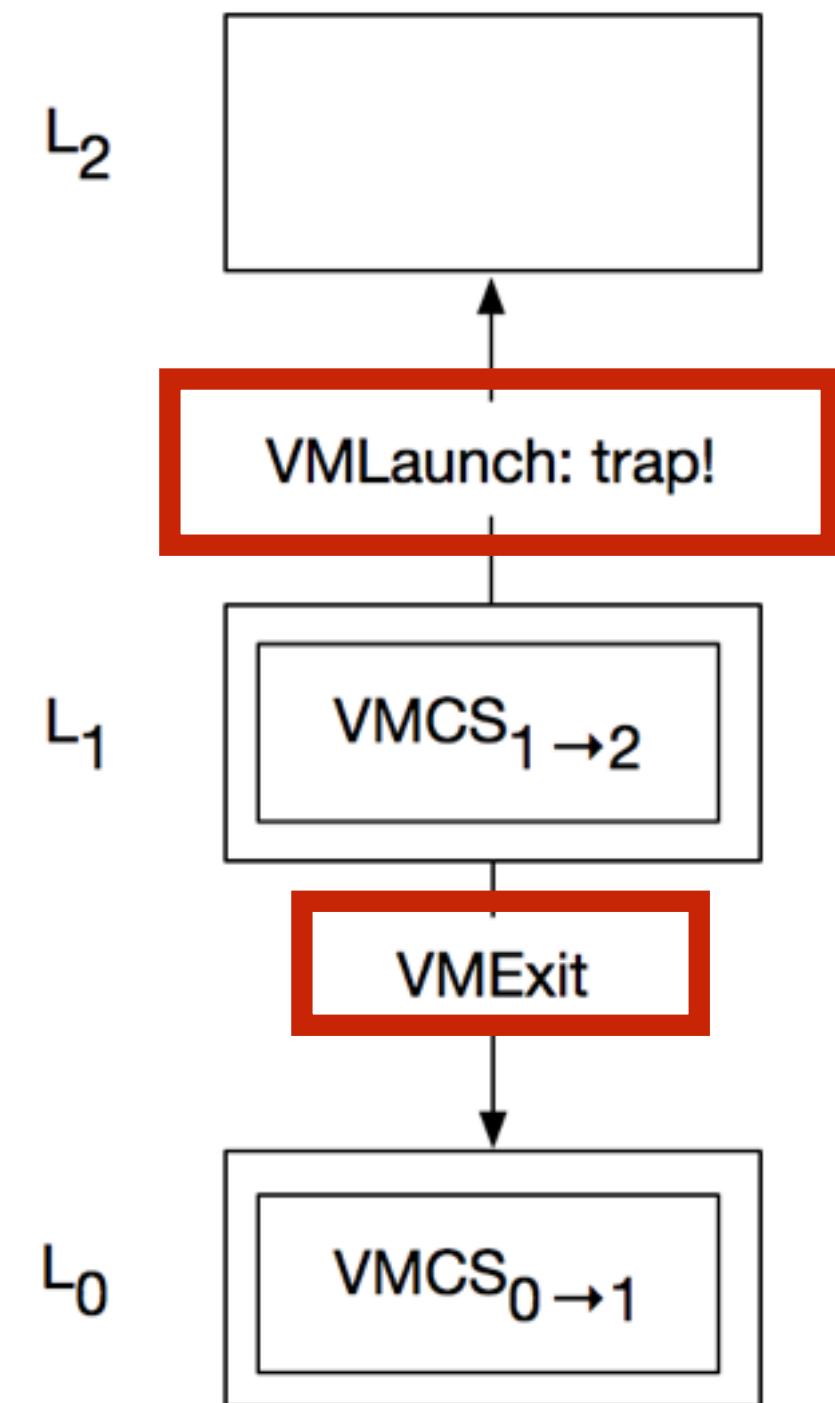
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$
- L_1 runs L_2 by creating $\text{VMCS}_{1 \rightarrow 2}$, then calls `vmlaunch`



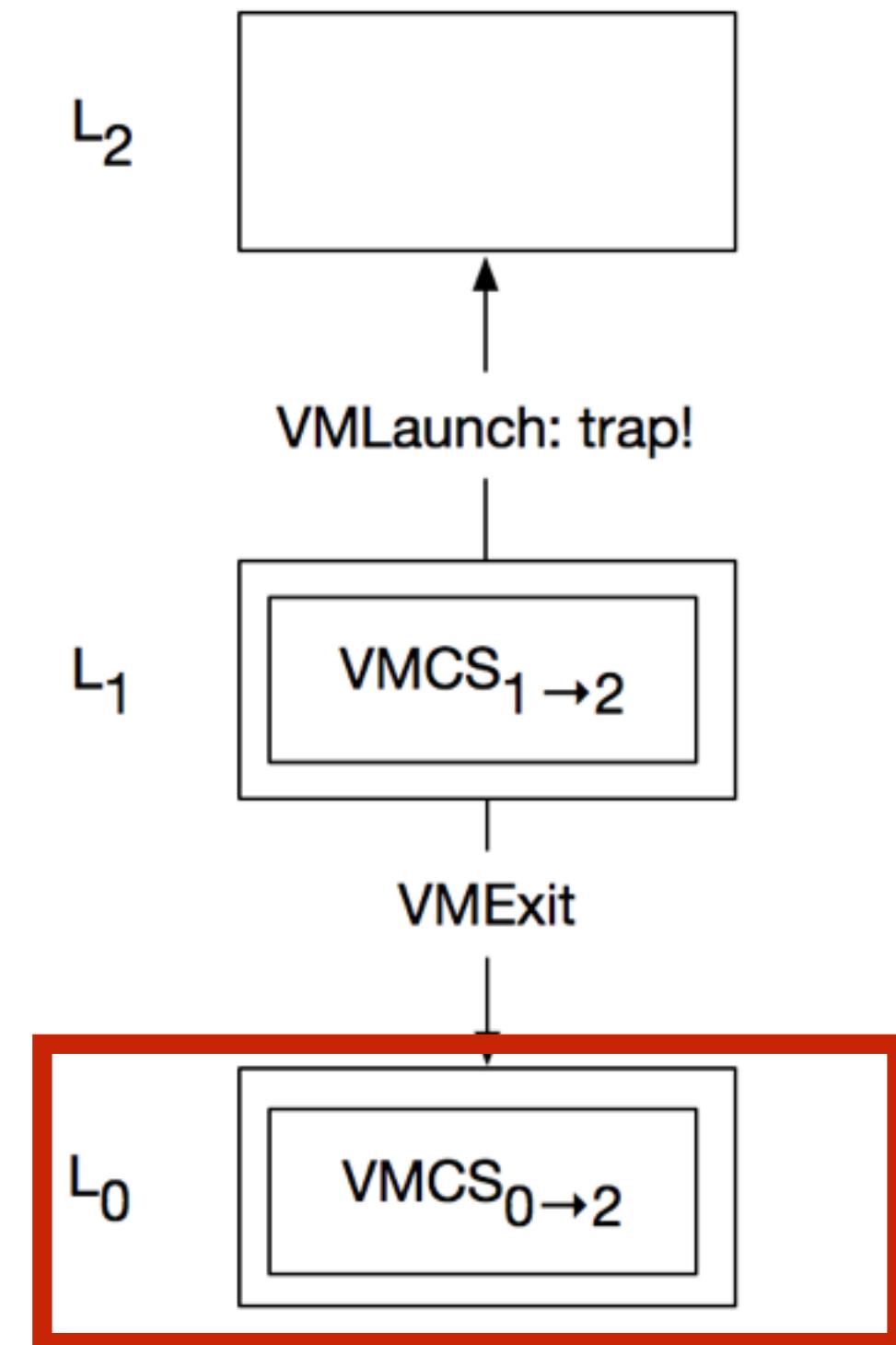
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$
- L_1 runs L_2 by creating $\text{VMCS}_{1 \rightarrow 2}$, then calls `vmlaunch`
- `vmlaunch` traps to L_0



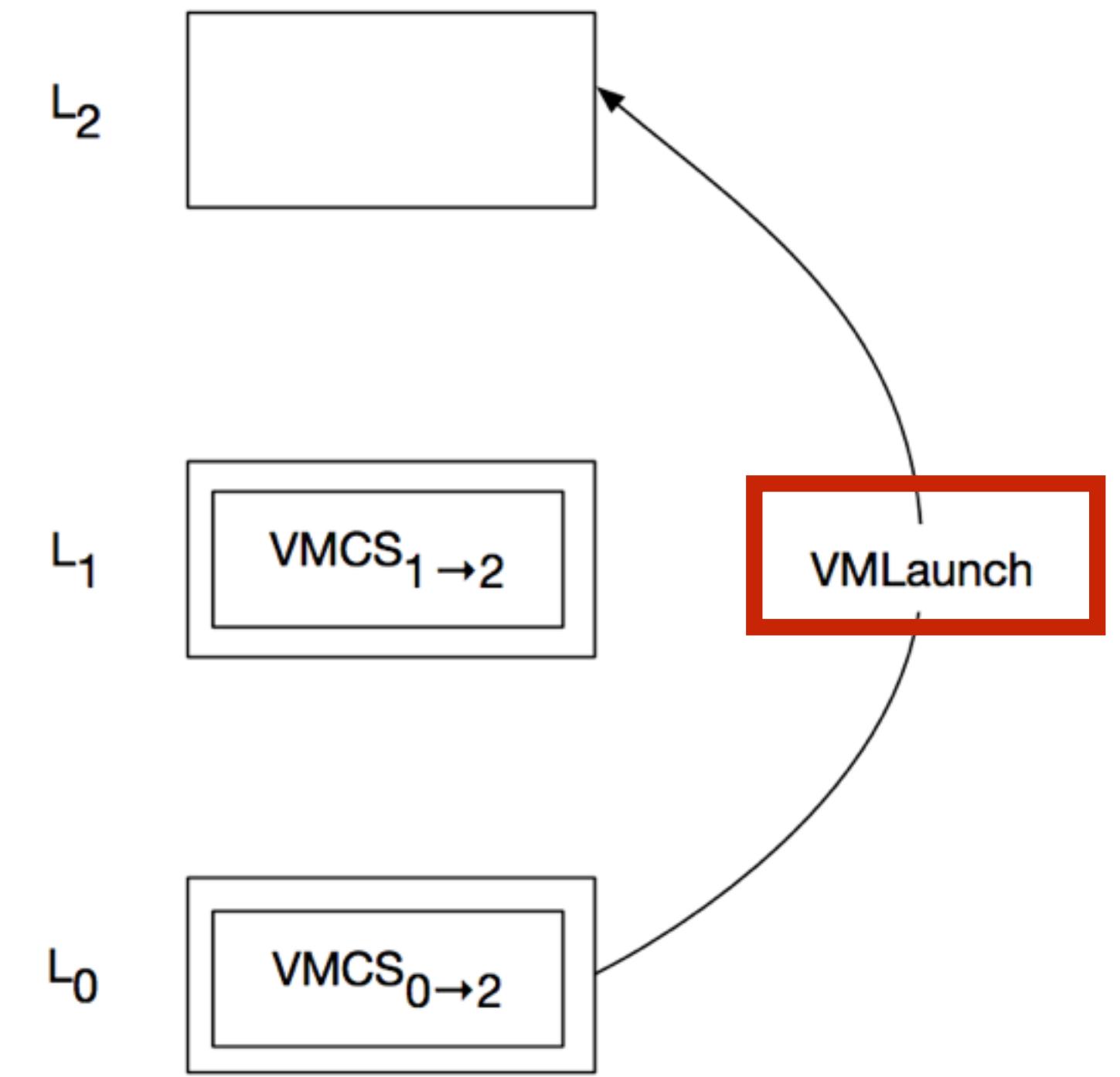
Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$
- L_1 runs L_2 by creating $\text{VMCS}_{1 \rightarrow 2}$, then calls `vmlaunch`
- `vmlaunch` traps to L_0
- L_0 merges $\text{VMCS}_{0 \rightarrow 1}$ and $\text{VMCS}_{1 \rightarrow 2}$ to create $\text{VMCS}_{0 \rightarrow 2}$



Nested CPU Virtualization: Launching a new VM

- L_0 runs L_1 using $\text{VMCS}_{0 \rightarrow 1}$
- L_1 runs L_2 by creating $\text{VMCS}_{1 \rightarrow 2}$, then calls `vmlaunch`
- `vmlaunch` traps to L_0
- L_0 merges $\text{VMCS}_{0 \rightarrow 1}$ and $\text{VMCS}_{1 \rightarrow 2}$ to create $\text{VMCS}_{0 \rightarrow 2}$
- L_0 launches L_2



Implementing Turtles

- Nested VMX Virtualization for CPUs
- **Multidimensional Paging for MMUs**
- Multi-level Device Assignment for I/O



MMU Virtualization

- With nested virtualization, we have to make n translations for n levels
 $L_2^{virtual} \rightarrow L_2^{physical}, L_2 \rightarrow L_1, L_1 \rightarrow L_0$



MMU Virtualization

- With nested virtualization, we have to make n translations for n levels
 $L_2^{virtual} \rightarrow L_2^{physical}, L_2 \rightarrow L_1, L_1 \rightarrow L_0$
- Unfortunately, only **two dimensions** of hardware support



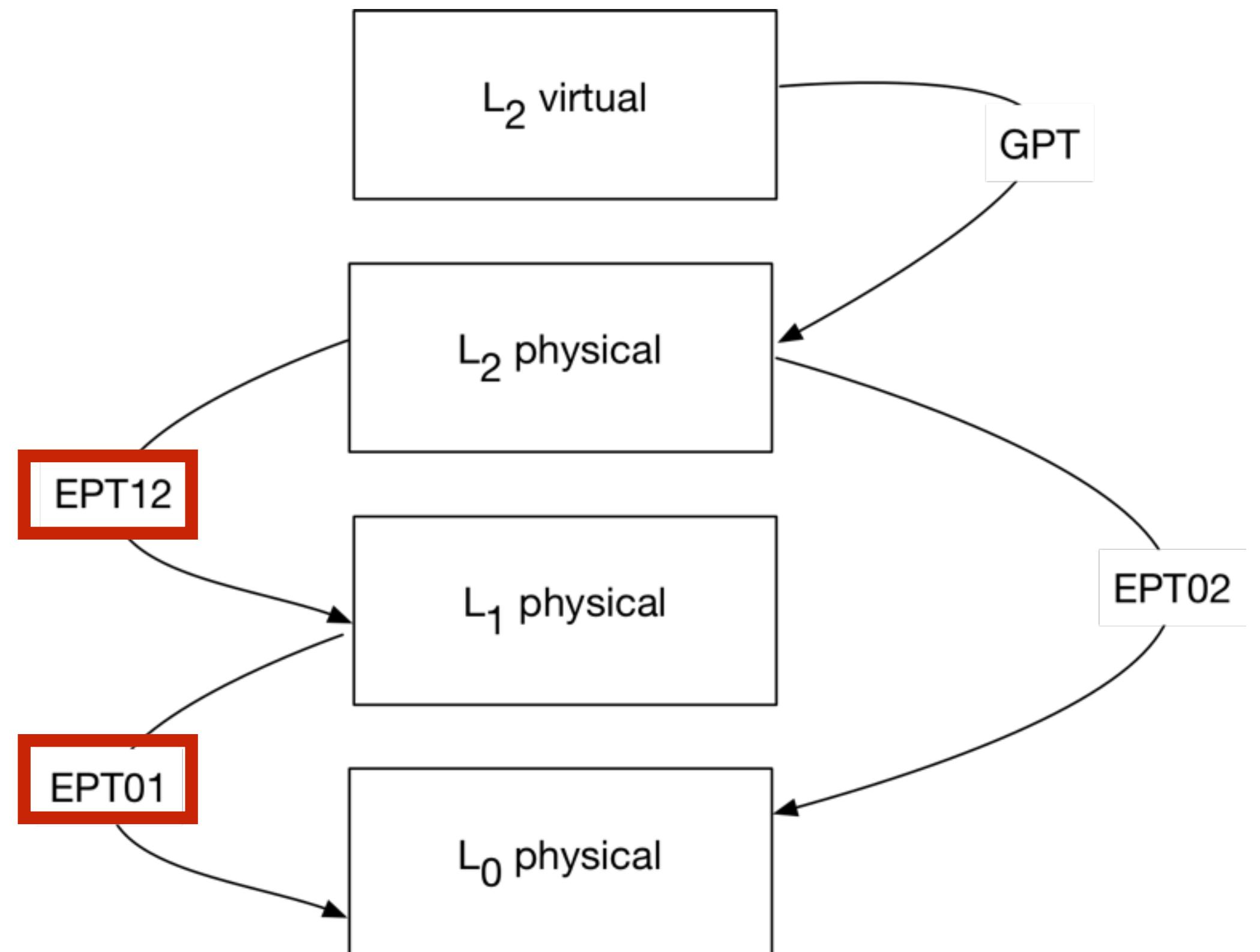
MMU Virtualization

- With nested virtualization, we have to make n translations for n levels
 $L_2^{virtual} \rightarrow L_2^{physical}, L_2 \rightarrow L_1, L_1 \rightarrow L_0$
- Unfortunately, only **two dimensions** of hardware support
 - EPT (extended page table) provides translation from guest physical to host physical



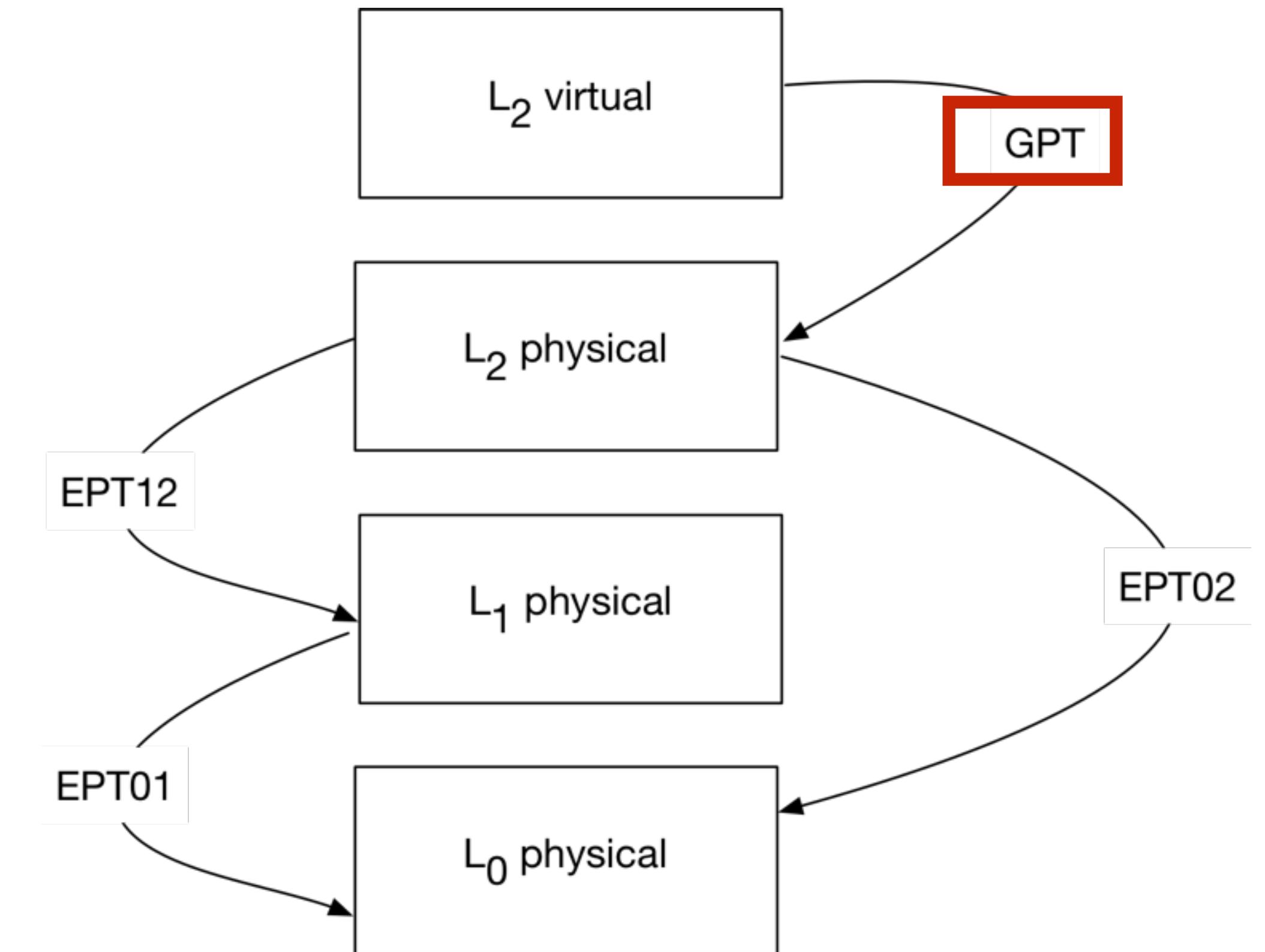
Multi-dimensional Paging

- Each level creates a virtual EPT that it exposes to higher levels



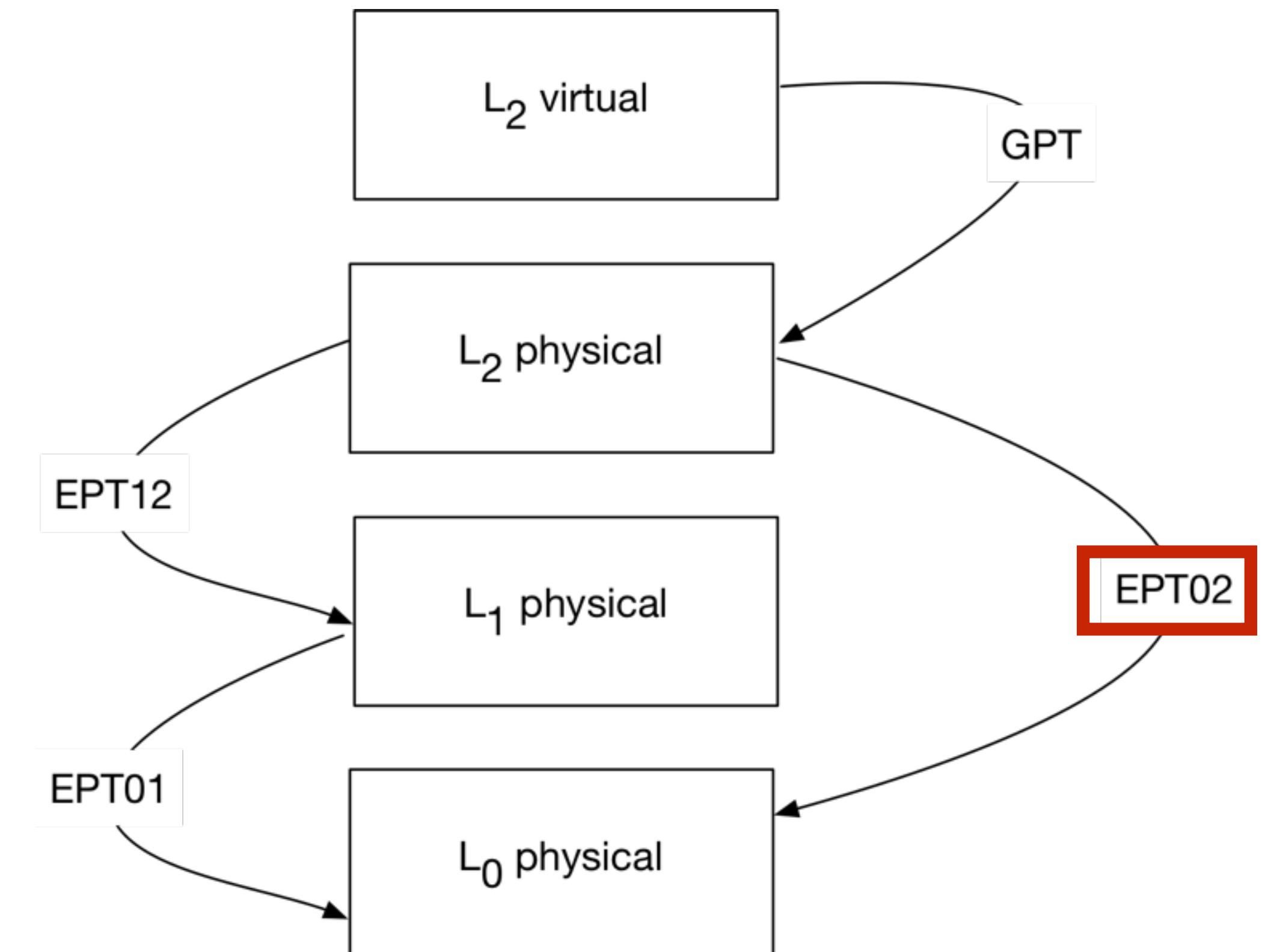
Multi-dimensional Paging

- Each level creates a virtual EPT that it exposes to higher levels
- Guest handles their own page table



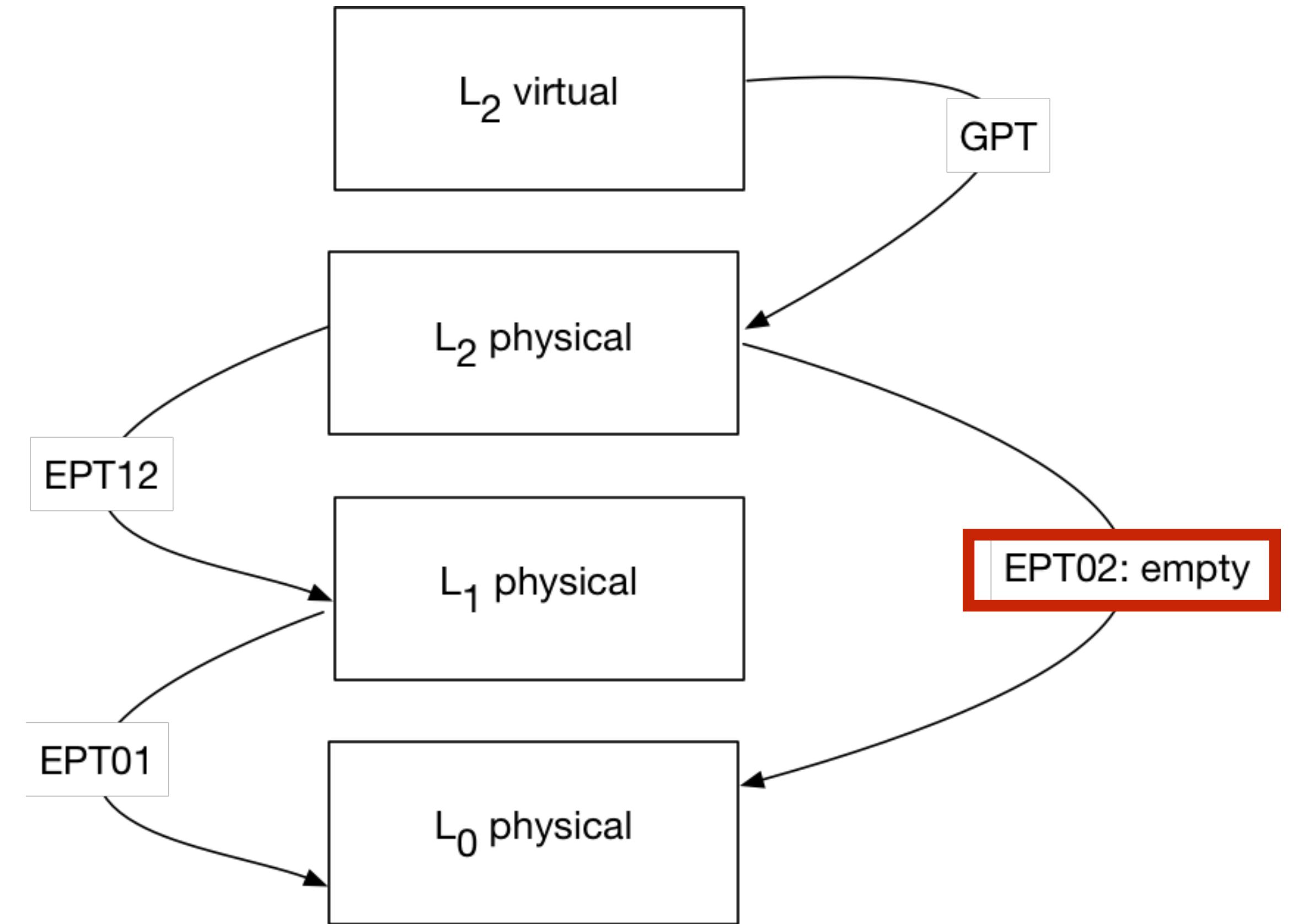
Multi-dimensional Paging

- Each level creates a virtual EPT that it exposes to higher levels
- Guest handles their own page table
- “Base” hypervisor translates address using *compressed* EPT



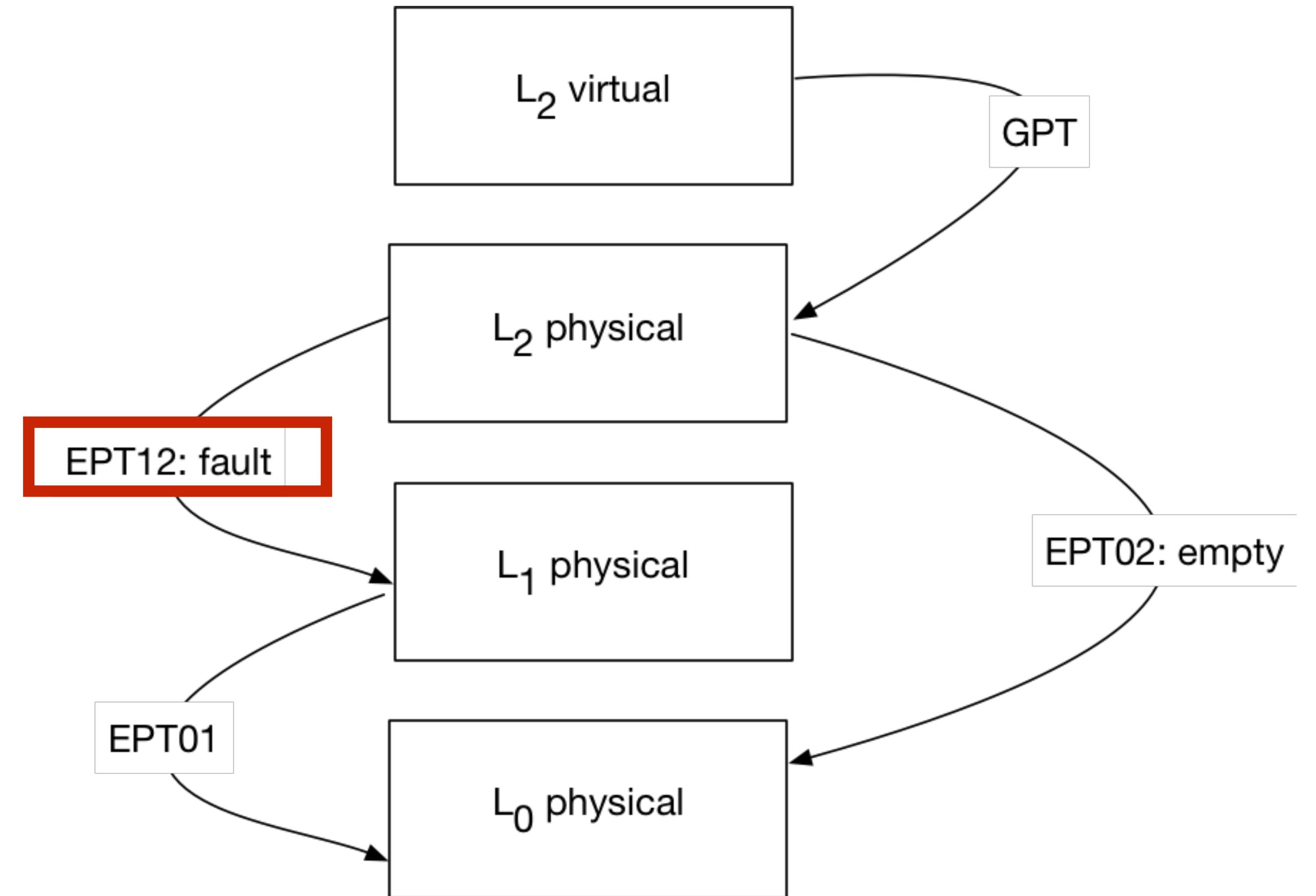
Building the Compressed EPT

- Start with empty $EPT_{0 \rightarrow 2}$



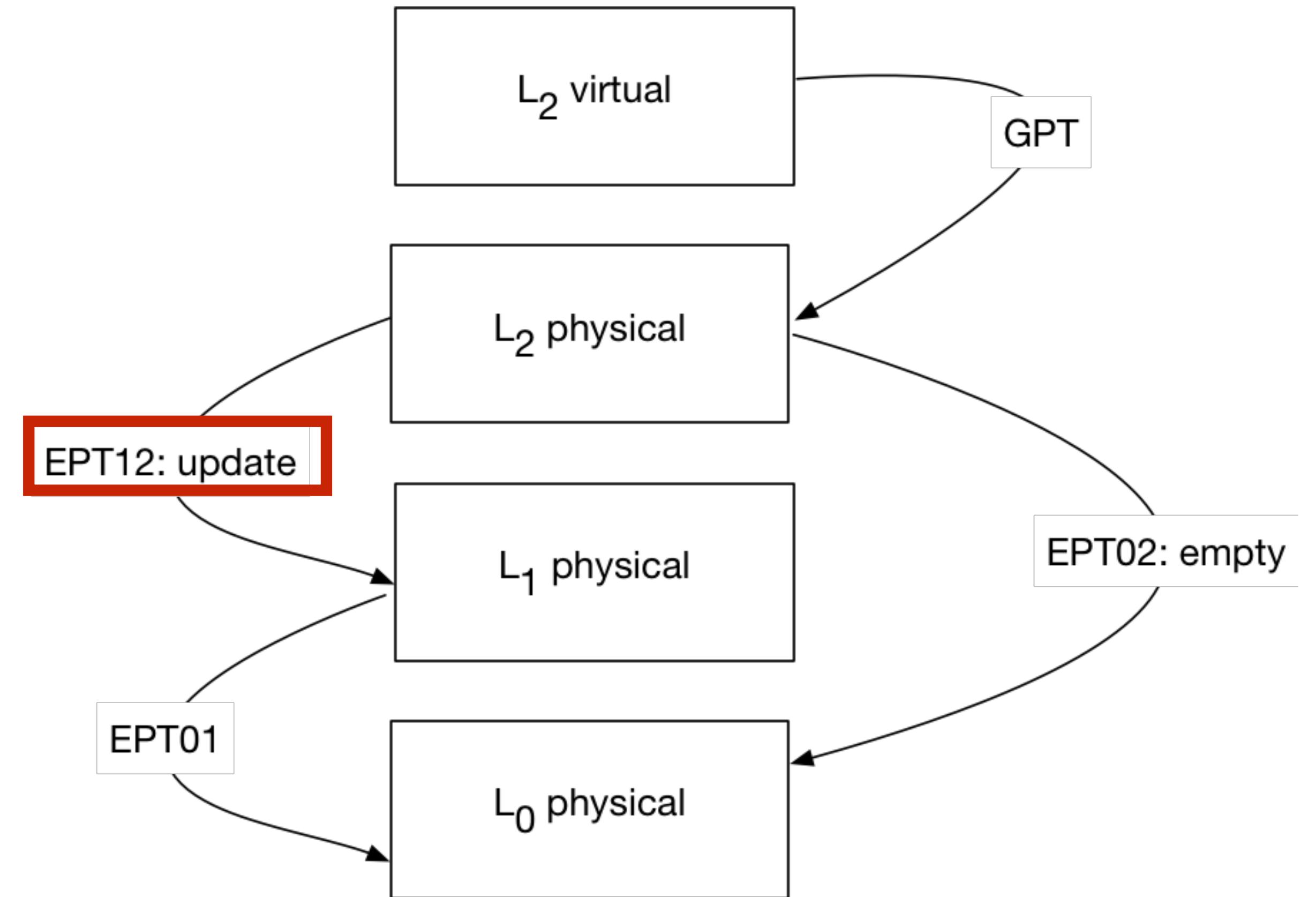
Building the Compressed EPT

- Start with empty $EPT_{0 \rightarrow 2}$
- When L_2 causes EPT-violation in $EPT_{1 \rightarrow 2}$



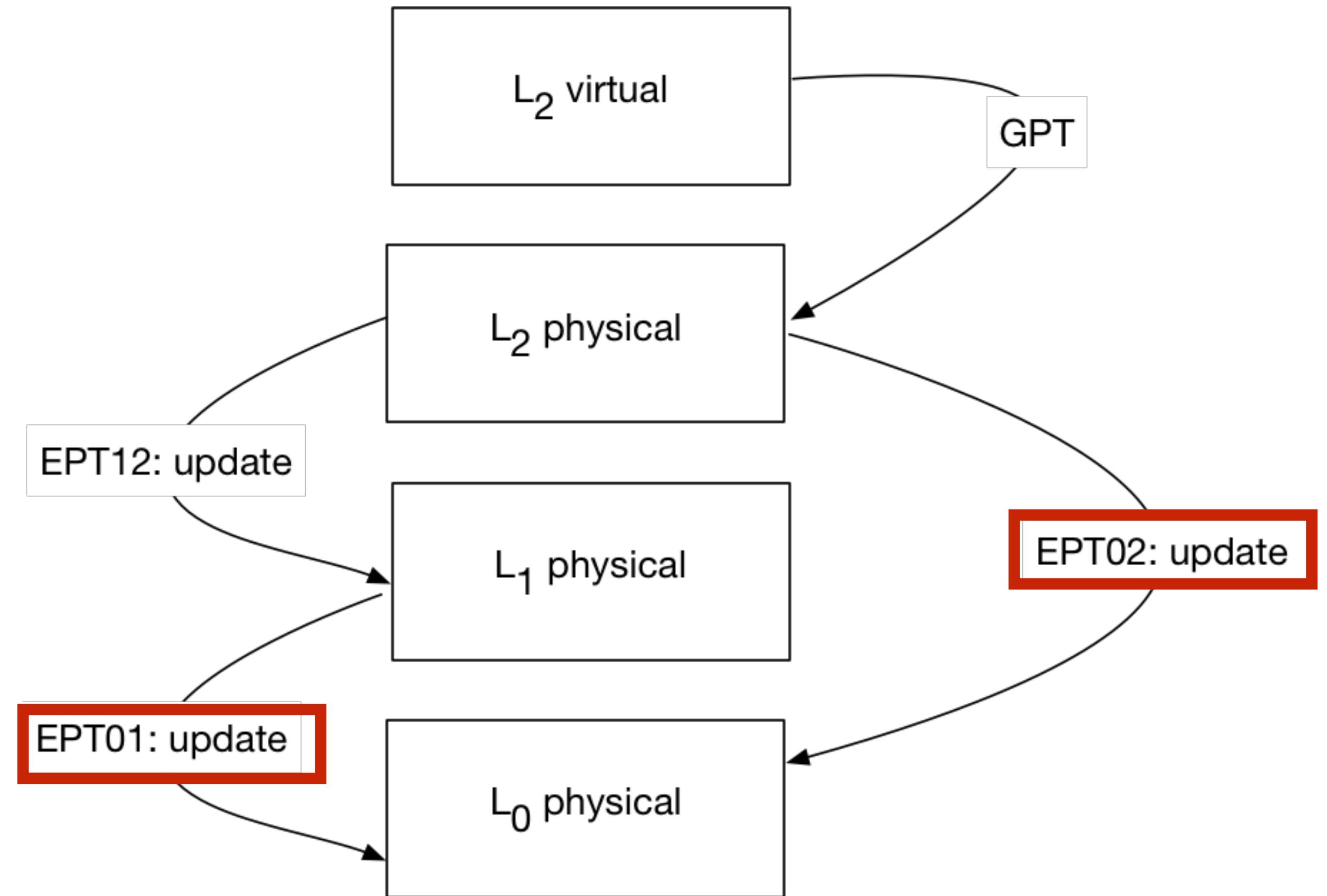
Building the Compressed EPT

- Start with empty $EPT_{0 \rightarrow 2}$
- When L_2 causes EPT-violation in $EPT_{1 \rightarrow 2}$
 - L_1 handles, updates $EPT_{1 \rightarrow 2}$



Building the Compressed EPT

- Start with empty $EPT_{0 \rightarrow 2}$
- When L_2 causes EPT-violation in $EPT_{1 \rightarrow 2}$
 - L_1 handles, updates $EPT_{1 \rightarrow 2}$
 - L_0 uses $EPT_{0 \rightarrow 1}$ and $EPT_{1 \rightarrow 2}$ to build $EPT_{0 \rightarrow 2}$



Implementing Turtles

- Nested VMX Virtualization for CPUs
- Multidimensional Paging for MMUs
- **Multi-level Device Assignment for I/O**



Multi-level device assignment

- I/O remains an outstanding challenge



Multi-level device assignment

- I/O remains an outstanding challenge
 - *Emulate* known devices



Multi-level device assignment

- I/O remains an outstanding challenge
 - *Emulate* known devices
 - Para-virtual drivers (hypercalls to base hypervisor)



Multi-level device assignment

- I/O remains an outstanding challenge
 - *Emulate* known devices
 - Para-virtual drivers (hypercalls to base hypervisor)
 - These are not performant enough!



Multi-level device assignment

- I/O remains an outstanding challenge
 - *Emulate* known devices
 - Para-virtual drivers (hypercalls to base hypervisor)
 - These are not performant enough!
- We have hardware support for virtual devices DMA via IOMMU
 - Translates guest physical addresses for I/O



Emulating the IOMMU

- L₀ emulates an IOMMU for L₁



Emulating the IOMMU

- L₀ emulates an IOMMU for L₁
- L₁ sets up IOMMU_{1→2} mapping



Emulating the IOMMU

- L₀ emulates an IOMMU for L₁
- L₁ sets up IOMMU_{1→2} mapping
- L₀ builds the IOMMU_{0→2} mapping



Emulating the IOMMU

- L₀ emulates an IOMMU for L₁
“It’s turtles all the way down!”
- L₁ sets up IOMMU_{2→1} mapping
- L₀ builds the IOMMU_{2→0} mapping



Optimizing Exit Handling

- Handling VMExits is slower in L₁ than in L₀
 - Updates to VMCS in exit handling is privileged
 - Solution: Trap the first time privileged exit code occurs, then replay in non-trapping environment
 - L₁ directly write to VMCS_{1→2}
 - Called DRW (direct read + write)



Overall Evaluation

- Two nested layers (L_1 and L_2)
- Macro and Microbenchmarks to identify general and worst cases
- KVM as base hypervisor (L_0)
 - Running KVM (L_1)
 - Running Ubuntu 9.04 (L_2)



Overall Evaluation

Kernbench

	Host	Guest	Nested	Nested _{DRW}
Runtime	324.3	355	406.3	391.5
% overhead vs. host	-	9.5	25.3	20.7
% overhead vs. guest	-	-	14.5	10.3
%CPU	93	97	99	99



Overall Evaluation

Kernbench

	Host	Guest	Nested	Nested _{DRW}
Runtime	324.3	355	406.3	391.5
% overhead vs. host	-	9.5	25.3	20.7
% overhead vs. guest	-	-	14.5	10.3
%CPU	93	97	99	99



Overall Evaluation

SPECjbb

	Host	Guest	Nested	Nested _{DRW}
Score	90493	83599	77065	78347
% degradation vs. host	-	7.6	14.8	13.4
% degradation vs. guest	-	-	7.8	6.3
%CPU	100	100	100	100



Overall Evaluation

SPECjbb

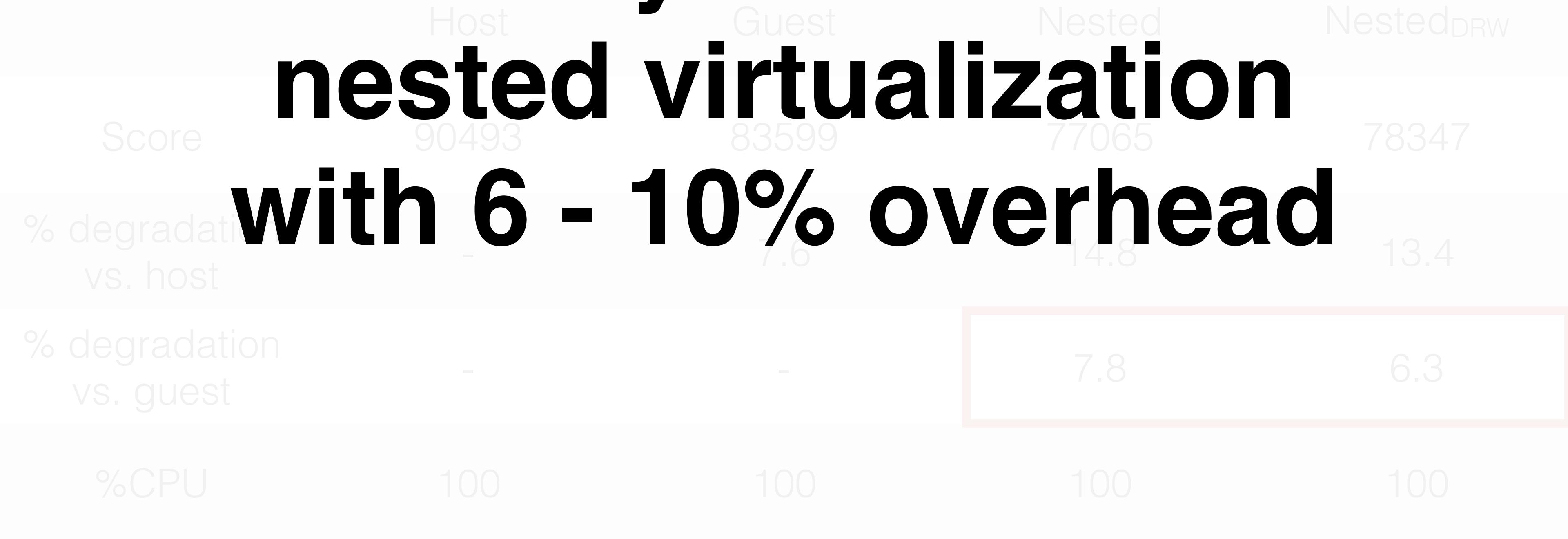
	Host	Guest	Nested	Nested _{DRW}
Score	90493	83599	77065	78347
% degradation vs. host	-	7.6	14.8	13.4
% degradation vs. guest	-	-	7.8	6.3
%CPU	100	100	100	100



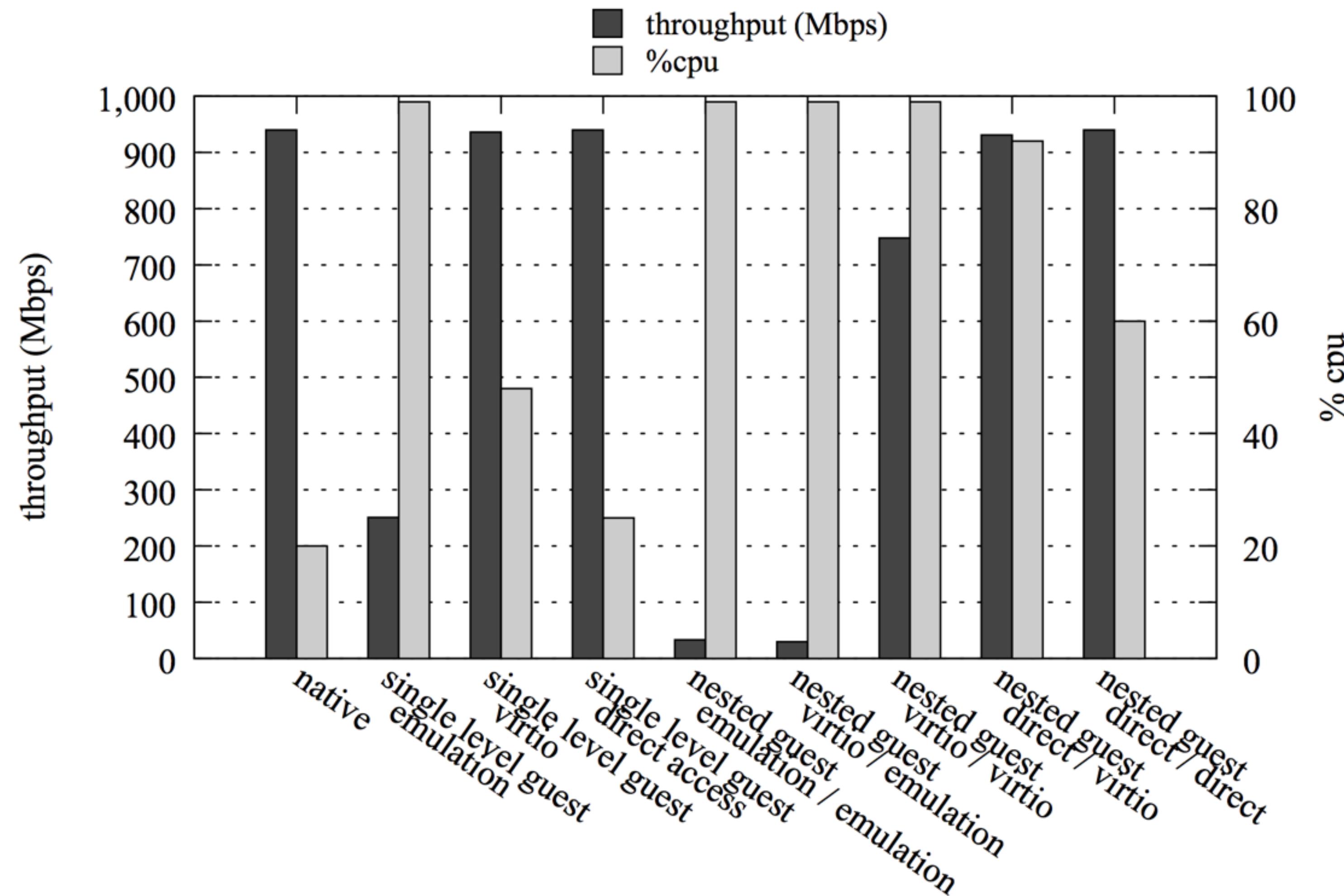
Overall Evaluation

**Takeaway: Can achieve
nested virtualization**

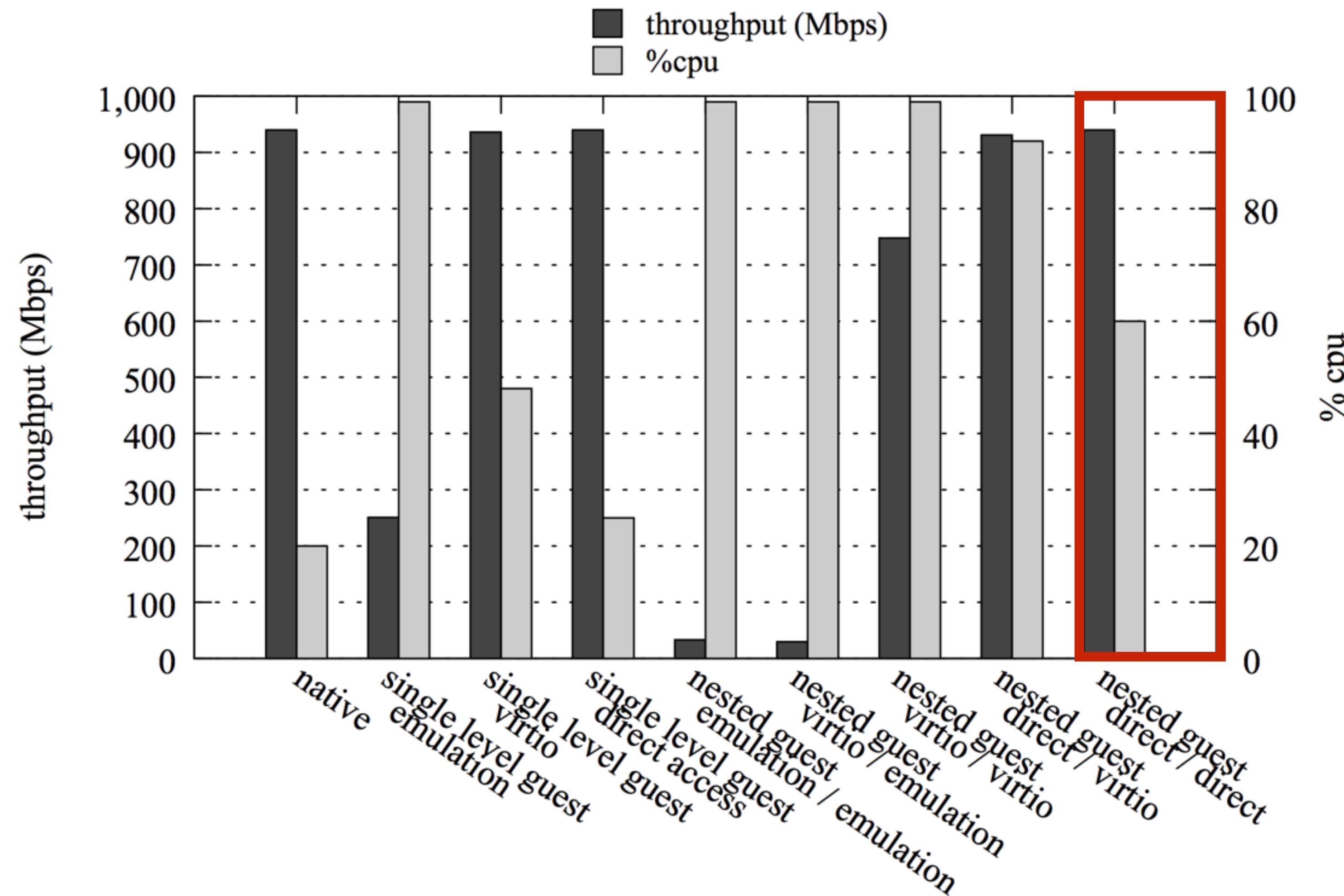
with 6 - 10% overhead



I/O Intensive Workloads

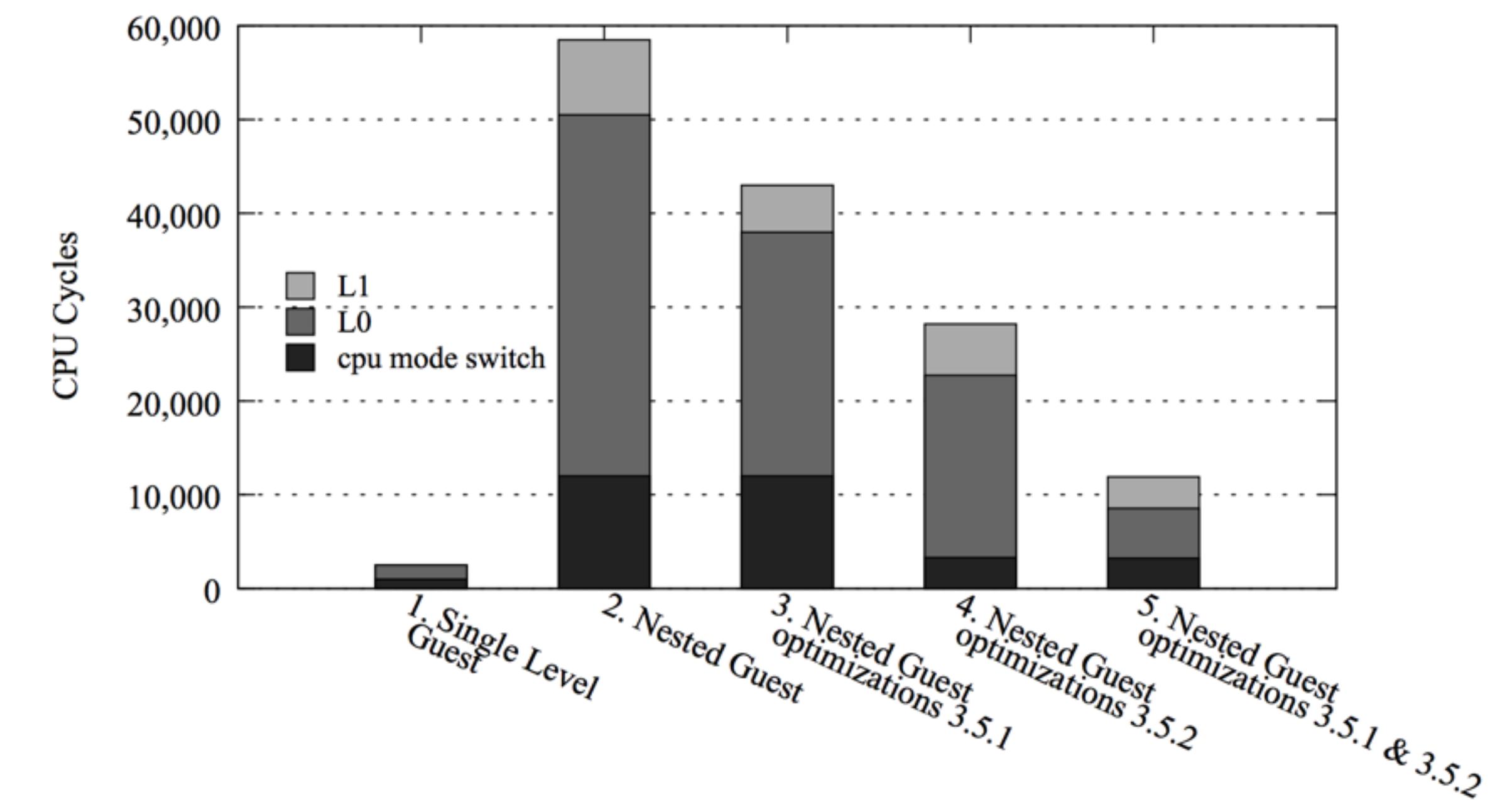


I/O Intensive Workloads



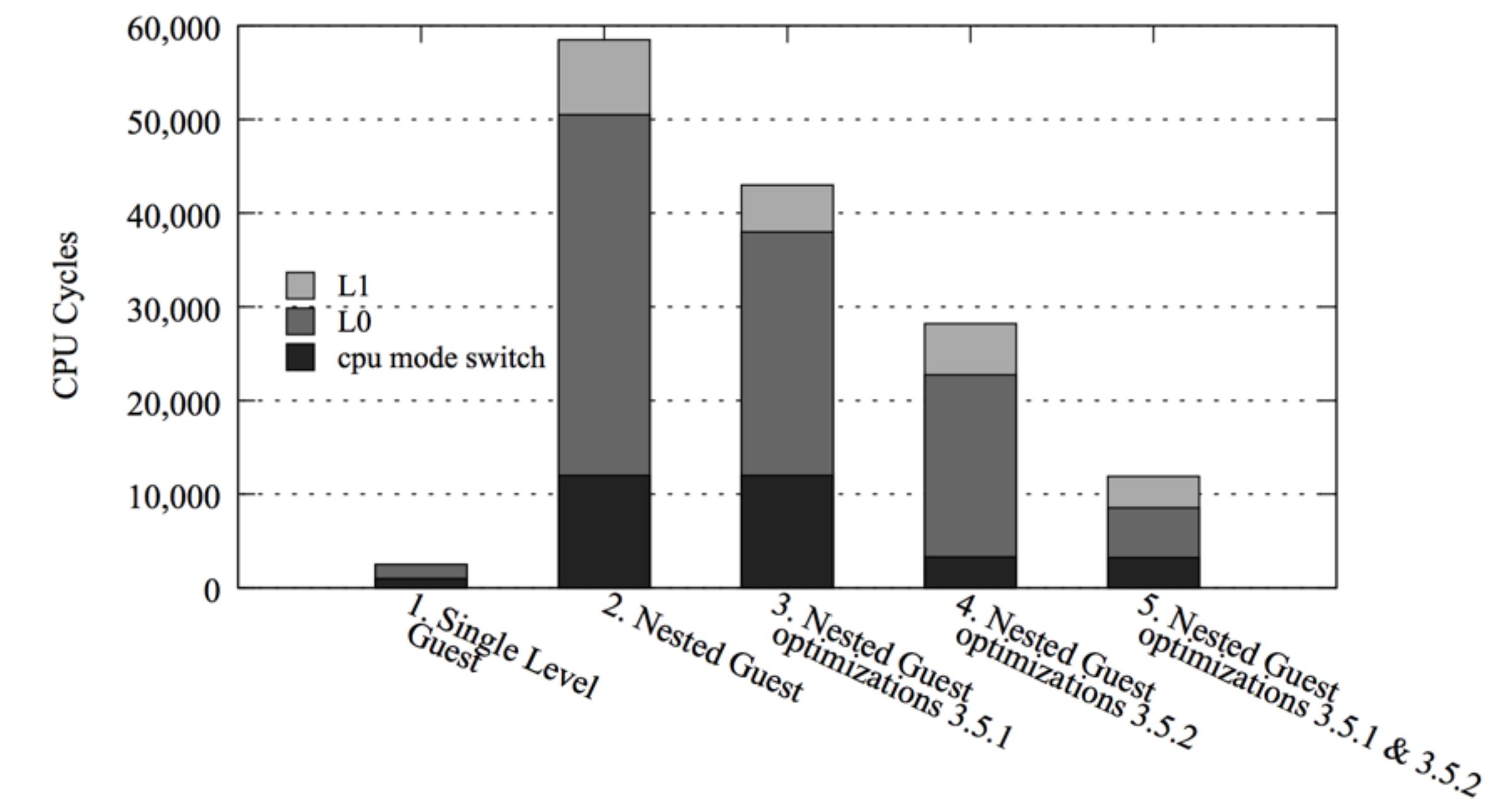
Microbenchmark

- Run “cpuid” in a loop to generate exits
 - Run 4M times



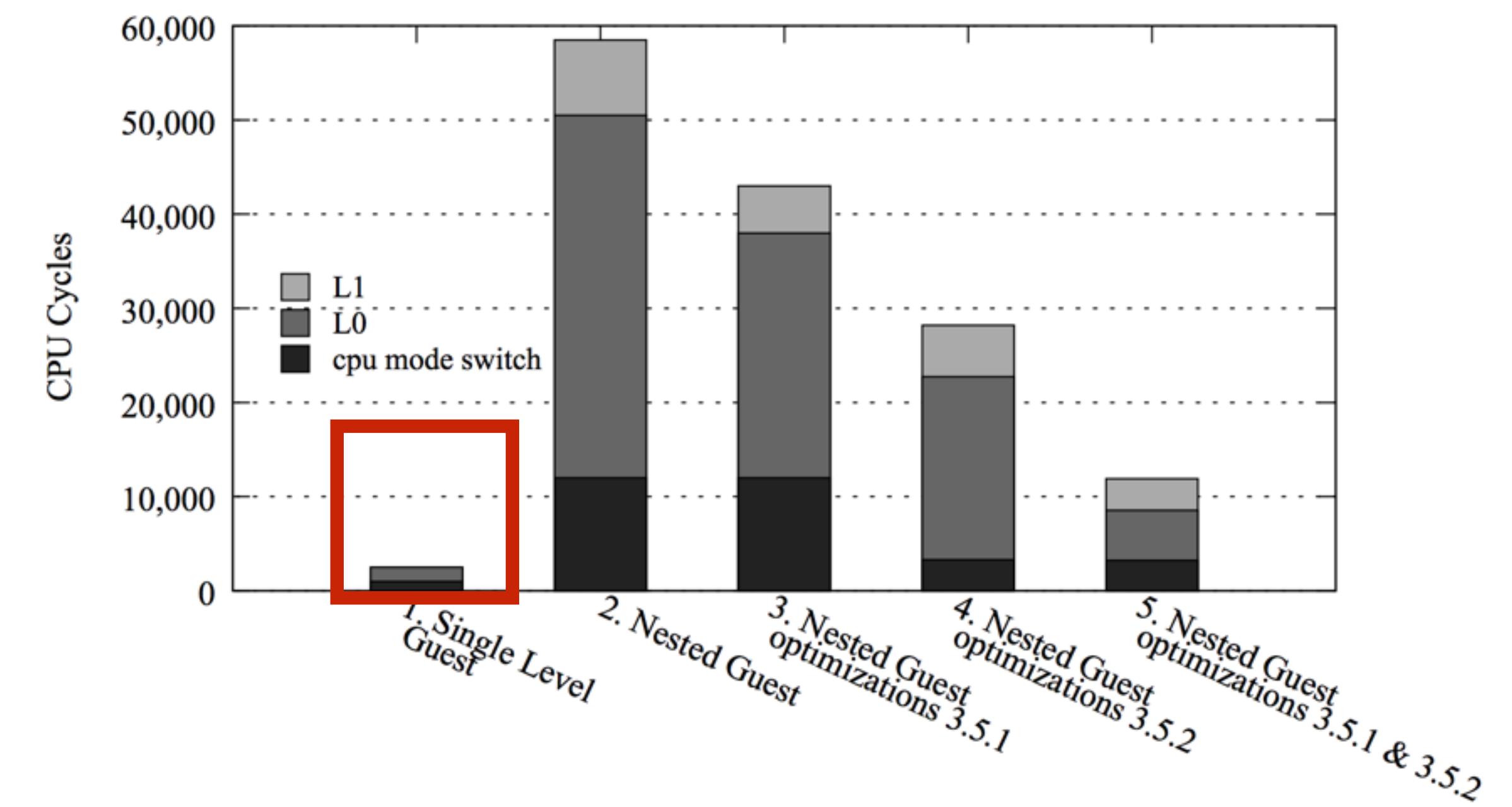
Microbenchmark

- Run “cpuid” in a loop to generate exits
 - Run 4M times
 - Bare metal: 100 cycles



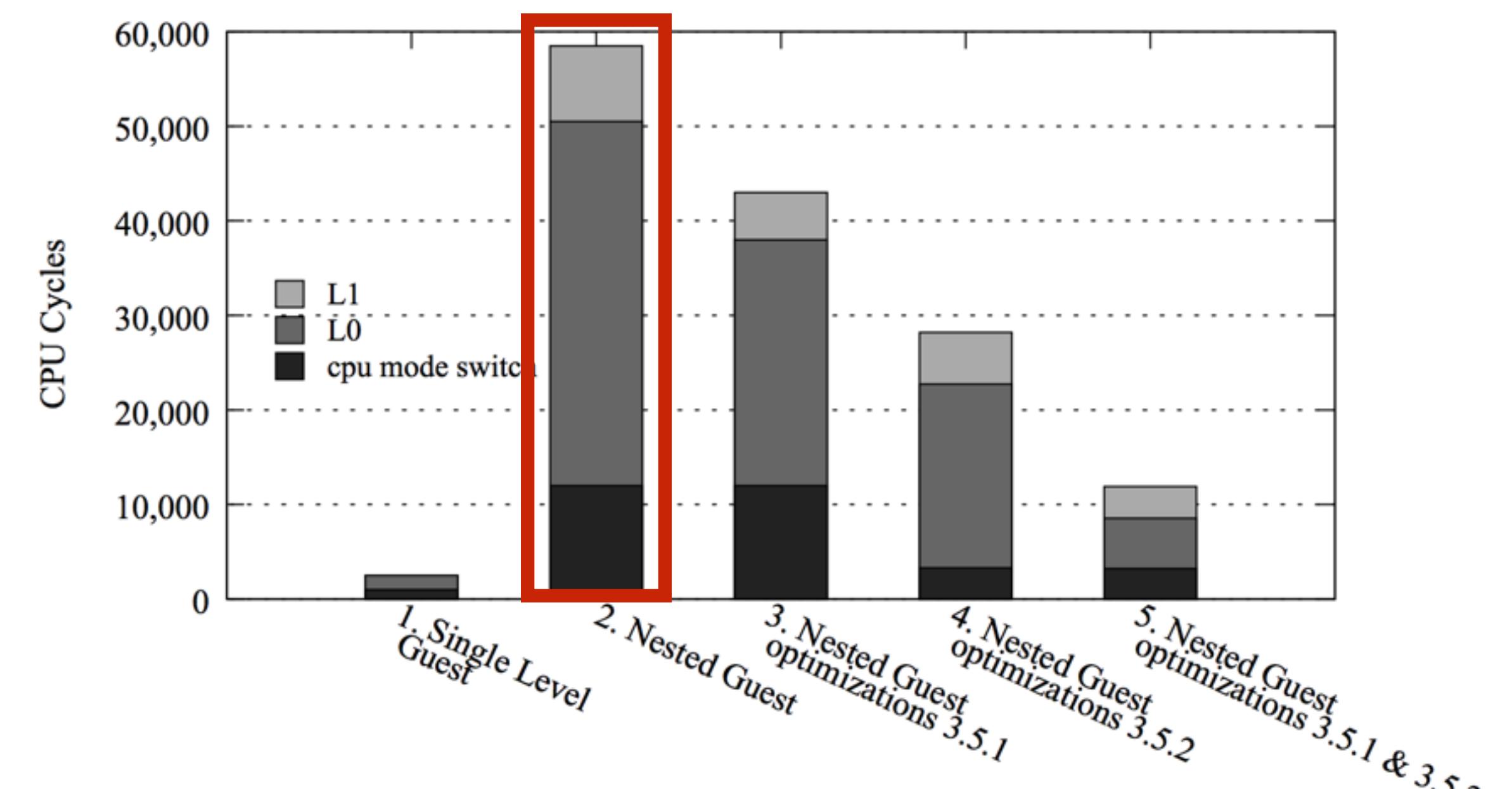
Microbenchmark

- Run “cpuid” in a loop to generate exits
 - Run 4M times
 - Bare metal: 100 cycles
 - Single level guest: 2,600 cycles



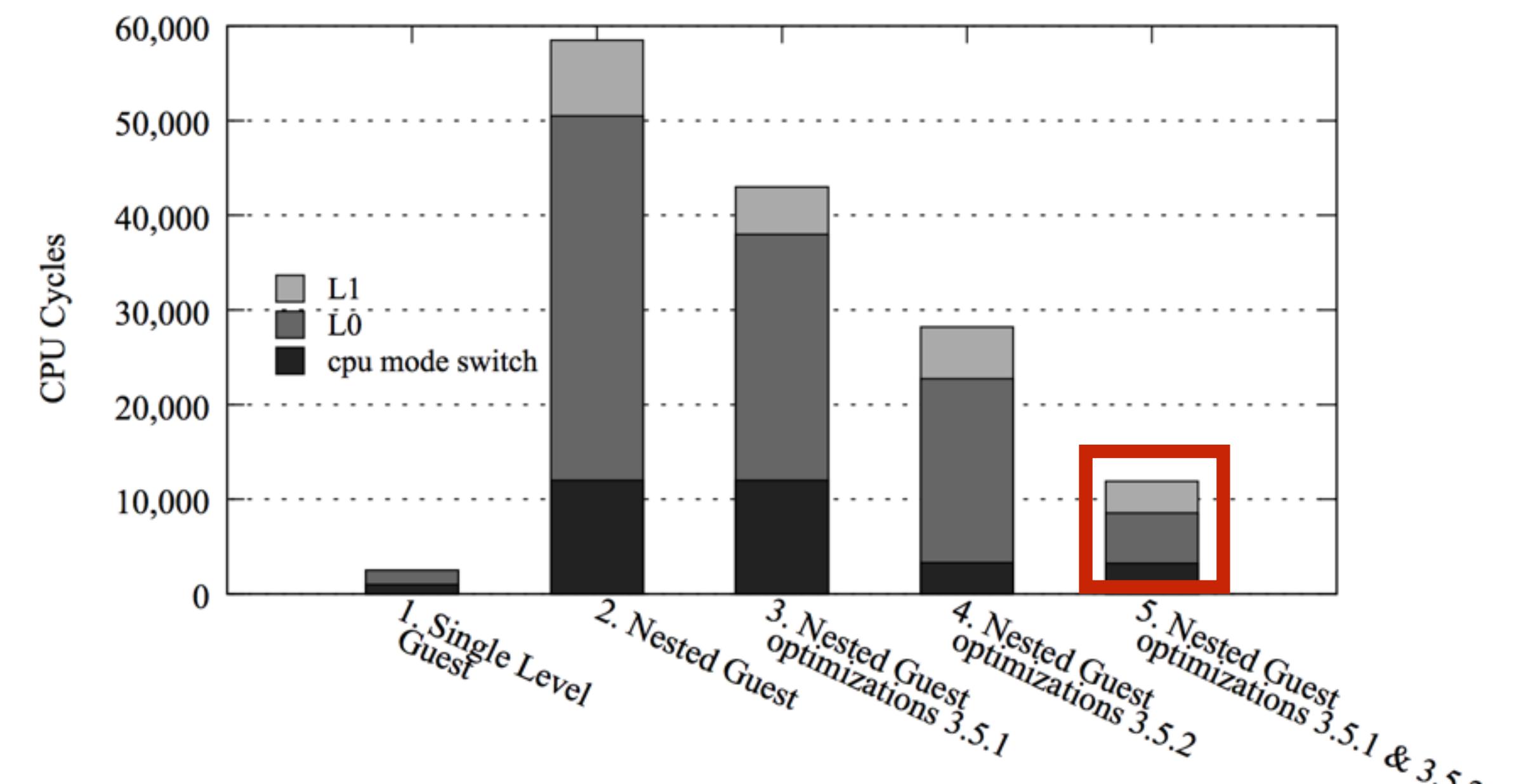
Microbenchmark

- Run “cpuid” in a loop to generate exits
 - Run 4M times
 - Bare metal: 100 cycles
 - Single level guest: 2,600 cycles
 - Nested guest: 58,000 cycles



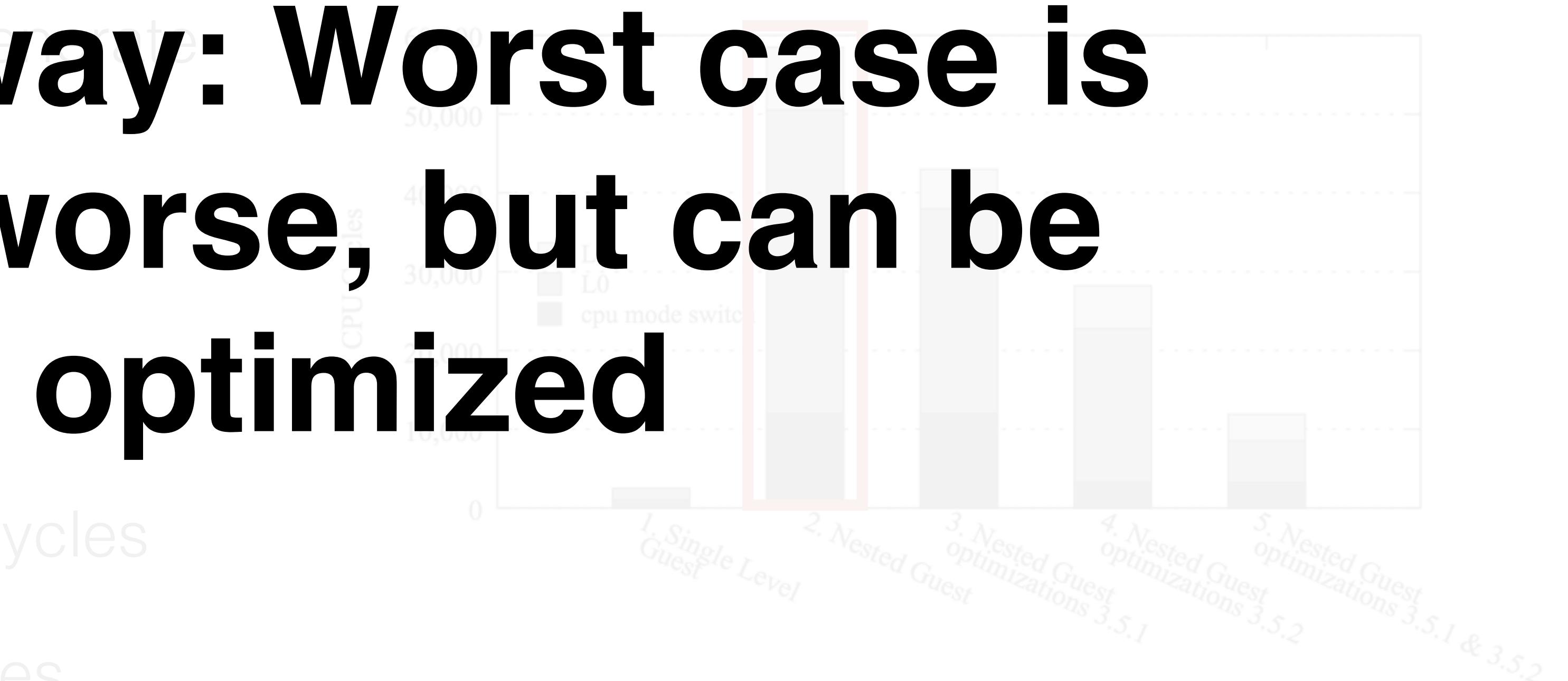
Microbenchmark

- Run “cpuid” in a loop to generate exits
 - Run 4M times
 - Bare metal: 100 cycles
 - Single level guest: 2,600 cycles
 - Nested guest: 58,000 cycles



Microbenchmark

- Run “cpuid” exits
- Run 4M times
- Bare metal: 100 cycles
- Single level guest: 2,600 cycles
- Nested guest: 58,000 cycles



My Takeaways

- “Turtles” provides a clean solution to a tough problem
 - In particular, beefing up the base hypervisor to handle all traps is an intuitive solution
 - However, software will always be slower than hardware



My Takeaways

- “Turtles” provides a clean solution to a tough problem
 - In particular, beefing up the base hypervisor to handle all traps is an intuitive solution
 - However, software will always be slower than hardware
- **Paper:** Theoretical lower bounds for virtualization



My Takeaways

- “Turtles” provides a clean solution to a tough problem
 - In particular, beefing up the base hypervisor to handle all traps is an intuitive solution
 - However, software will always be slower than hardware
- **Paper:** Theoretical lower bounds for virtualization
- Nested virtualization is not practical for all workloads
 - I/O intensive workloads have unreasonable costs (60% vs 20% CPU overhead)
 - Largely because of trap-and-emulate, which requires VM + Host to share the same core



My Takeaways

- “Turtles” provides a clean solution to a tough problem
 - In particular, beefing up the base hypervisor to handle all traps is an intuitive solution
 - However, software will always be slower than hardware
- **Paper:** Theoretical lower bounds for virtualization
- Nested virtualization is not practical for all workloads
 - I/O intensive workloads have unreasonable costs (60% vs 20%)
 - Largely because of trap-and-emulate, which requires VM + Host to share the same core
- **Paper:** Rebuilding trap-and-emulate for I/O intensive workloads on multicore



References

- [1] <https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/>

