Reducing The Graphical User Interface's Consumption of Power on Mobile Devices

Brad Vander Zanden and Wei Gao

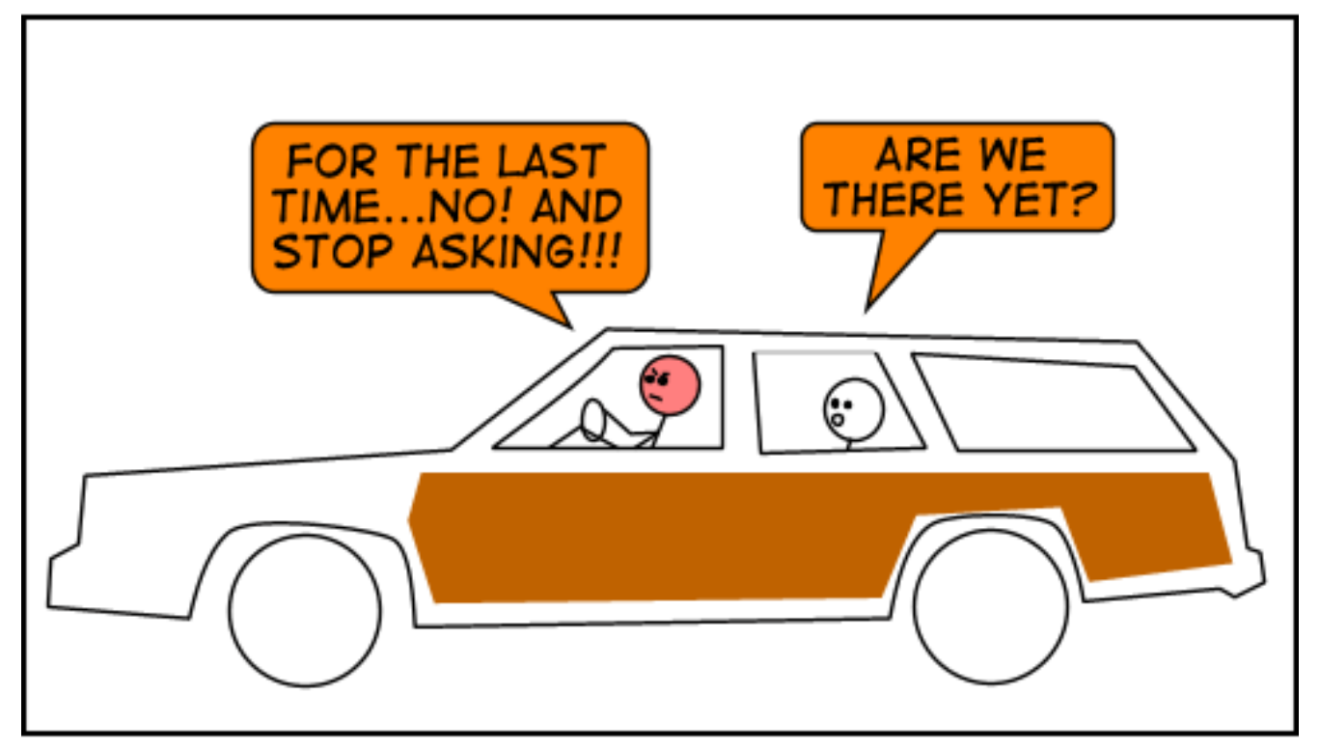
Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville

Award No. 1617198

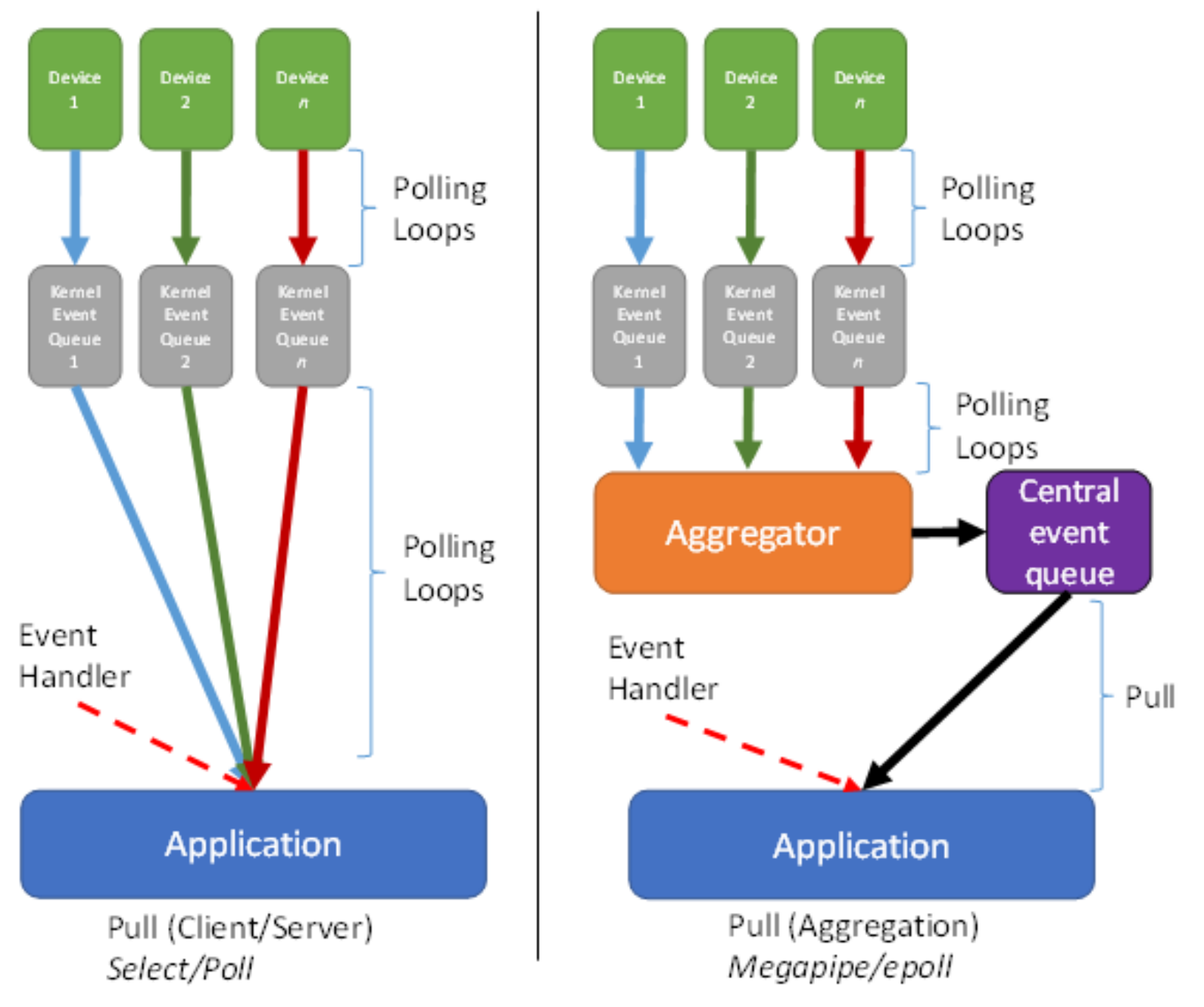
ABSTRACT

- A mobile device must be recharged on a daily basis even if it was in the user's pocket all day![3]
- Many apps needlessly drain the battery when you are not interacting with your device and prevent the CPU from being placed in energy-conserving "sleep" states[1].
- Problem Source: Mobile OS's use an event handling "pull" model, where an application must constantly poll to see if events are present, thus keeping the CPU aroused[2].
- Problem Solution: We are developing a "push" model, where the CPU sleeps until events occur—then the OS awakens the appropriate app and executes it.
- Preliminary results with the GUI portion of a mobile app shows large power savings and a reduction in latency (the time between an event's occurrence and an application's response)

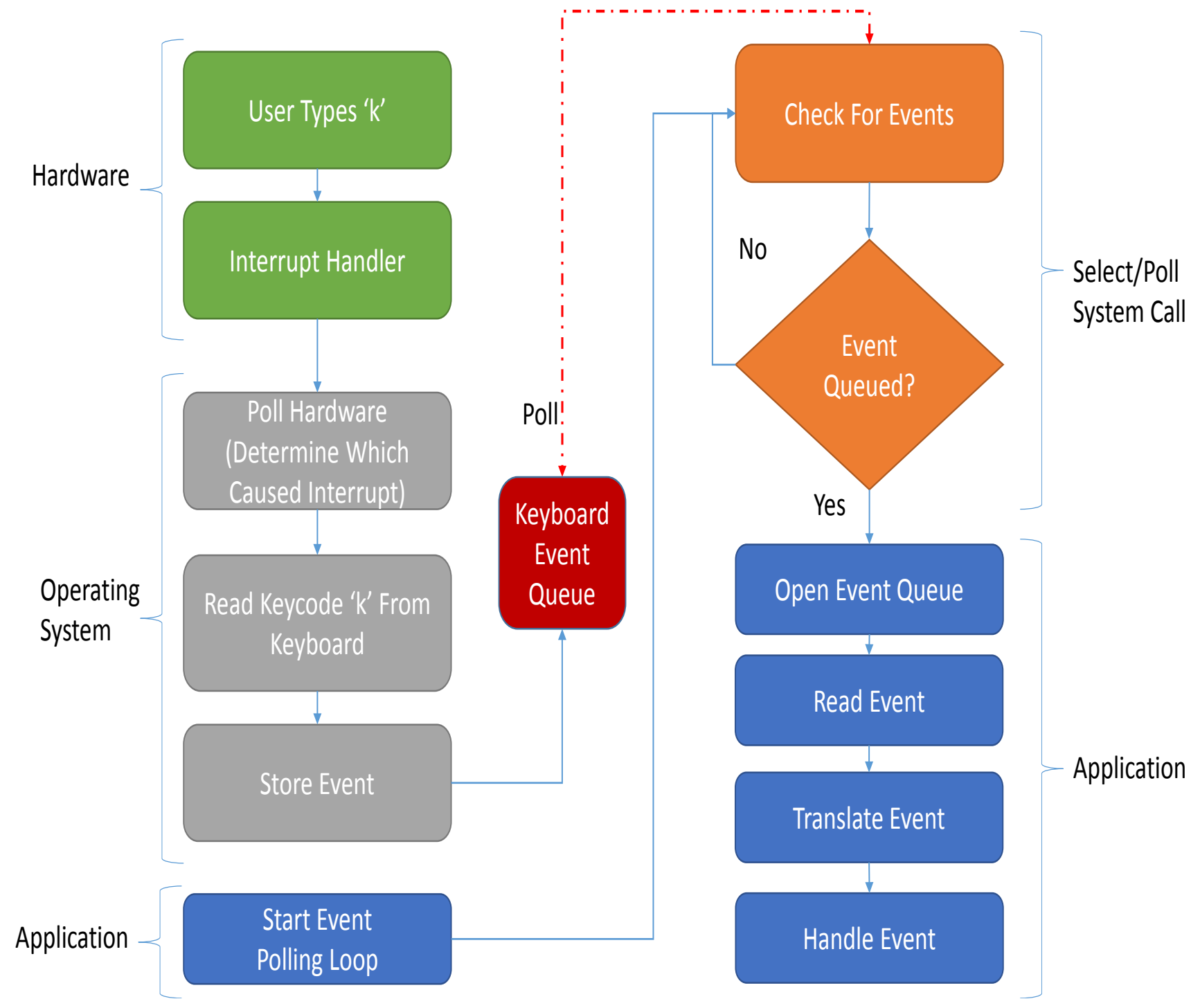
MOTIVATION: PROBLEM OF THE PULL MODEL



Power consumption in an idle, mobile device can be attributed largely to an event model known as the "pull" model, which repeatedly queries the operating system and hardware to determine if events exist.



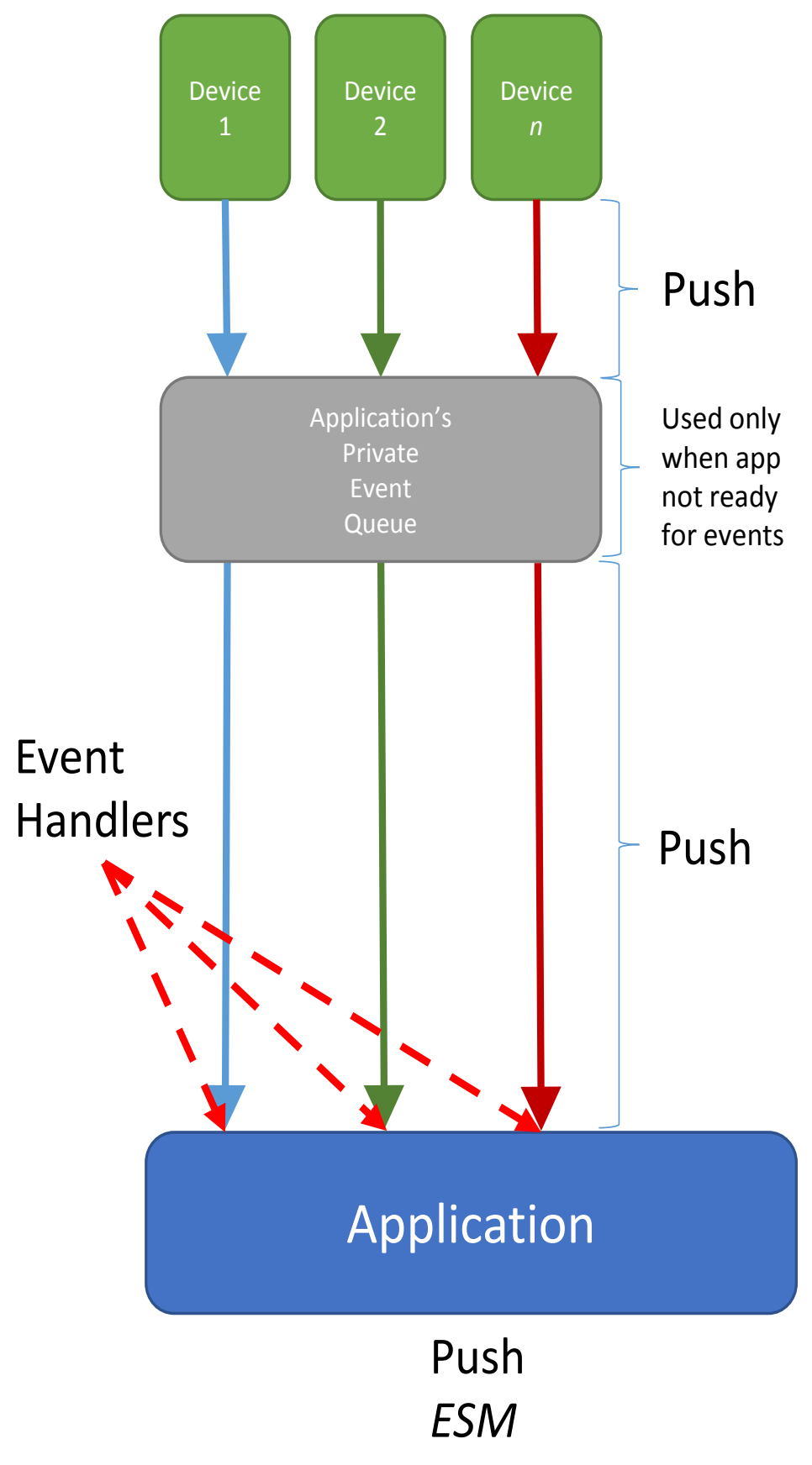
*Pull models implemented in the Android operating system all utilize a polling loop to continuously check for the existence of an event[5]. **Left:** Original pull model. **Right:** Improved pull model with event aggregation. CPU must still poll individual devices but application only polls one event queue, rather than multiple event queues[4].*

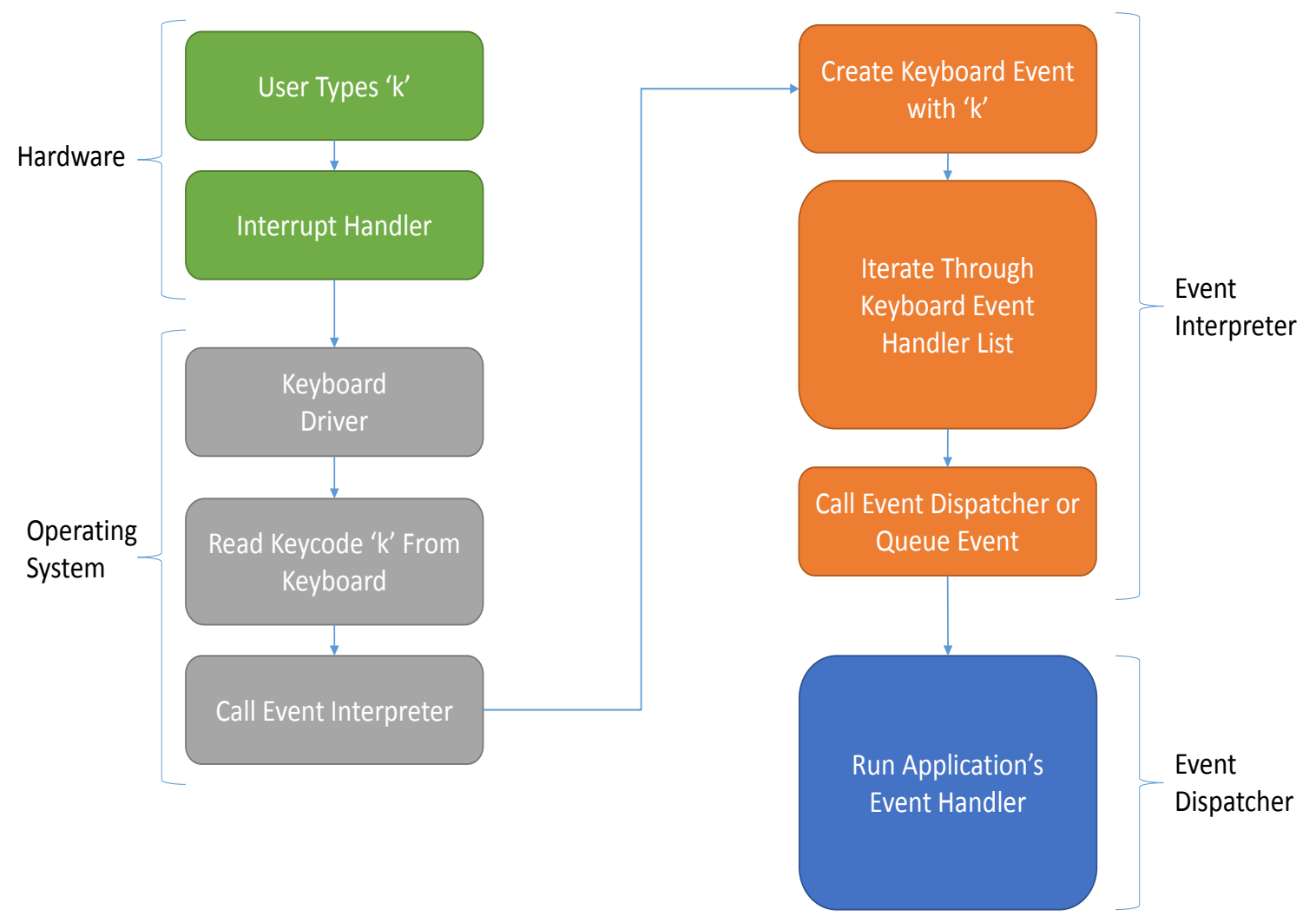


The flow of the current pull model shows how a keyboard input event, such as the user typing 'k', is handled by the operating system, and in the end, by the application.

RESEARCH APPROACHES

- Take advantage of new mobile device hardware that allows
  - Devices to "push" events to OS when events occurs
  - CPU to enter "deep sleep" states that conserve power
  - tasks to be distributed to multiple cores
- Use the Android OS and NVIDIA TK1 for benchmarking and prototyping

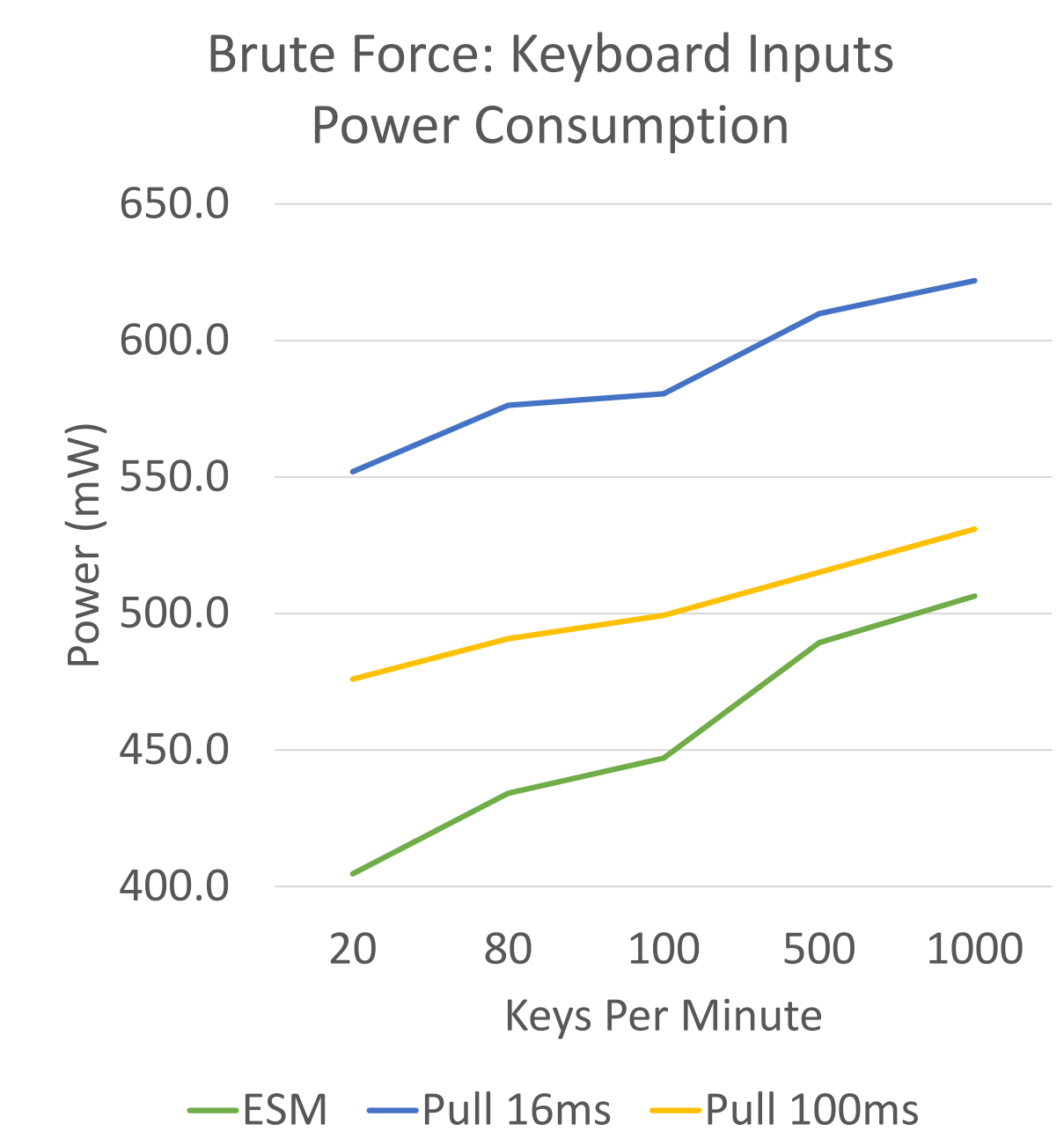

Device 1
Device 2
Device n
Push
Application's Private Event Queue
Used only when app not ready for events
Push
Event Handlers
Application
Push
ESM


Hardware
User Types 'k'
Interrupt Handler
Operating System
Keyboard Driver
Read Keycode 'k' From Keyboard
Call Event Interpreter
Create Keyboard Event with 'k'
Iterate Through Keyboard Event Handler List
Call Event Dispatcher or Queue Event
Event Interpreter
Run Application's Event Handler
Event Dispatcher

Our "push" Event Stream Model (ESM):

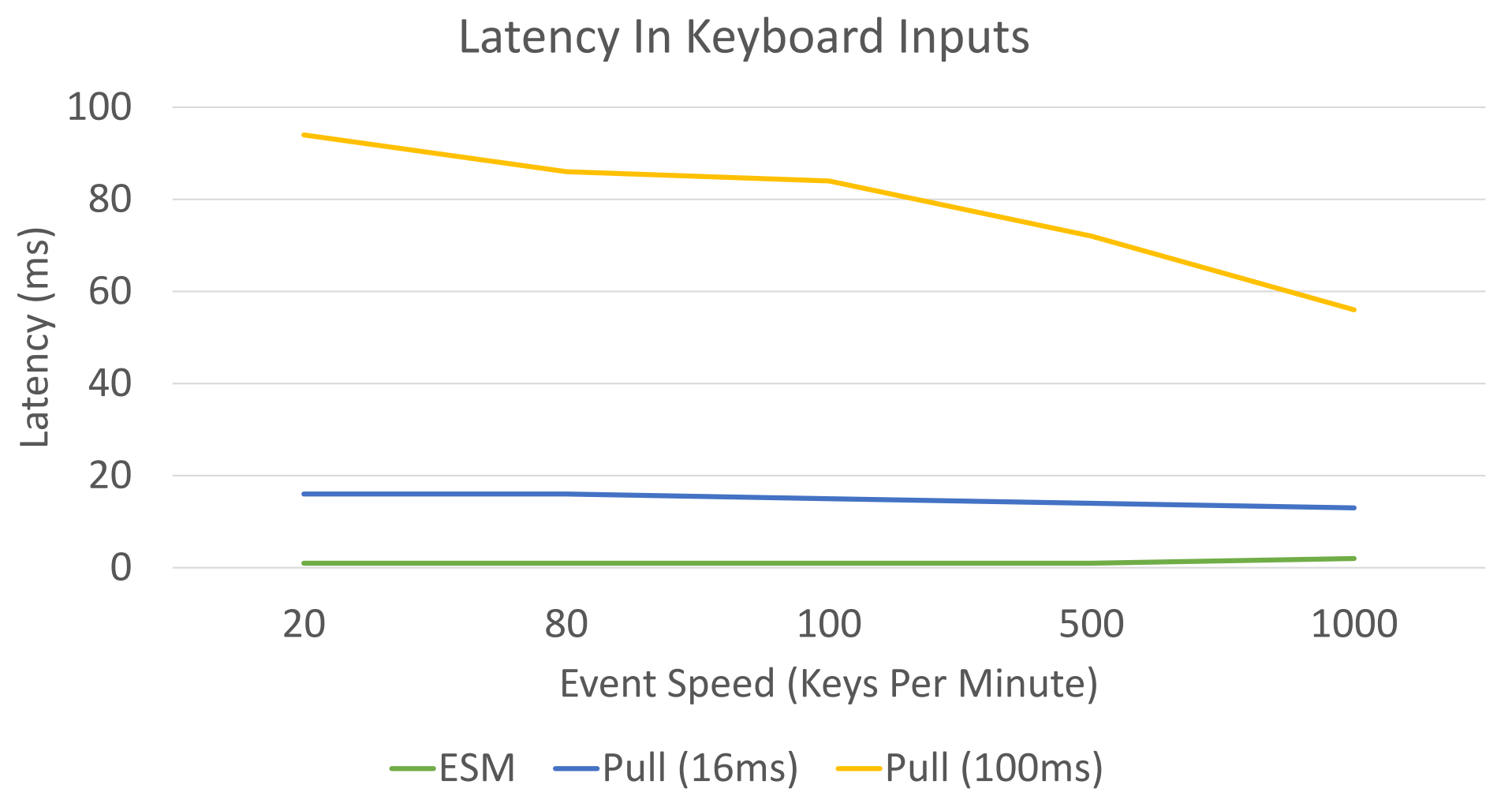
1. *Eliminates polling: Vectored interrupt controllers awaken CPU when event occurs*
2. *Streamlines event-handling*
   a. *Keyboard driver: Helps OS obtain information about the event*
   b. *Event interpreter: Determine which apps want the event*
   c. *Event dispatcher: Runs event handler that app has registered*
3. *Allows CPU to sleep when event system is dormant*

CURRENT PROGRESS



- ✓ *ESM has been implemented for the keyboard and mouse event portions of Android's OS*
- ✓ *Versus a pull model with a 16 millisecond delay (current state of the art for Android OS), ESM reduces power consumption by*
  - ✓ *~250mW for infrequent events*
  - ✓ *~120mW for frequent events.*
- ✓ *Versus a pull model with a 100 millisecond poll loop, ESM reduces power consumption by*
  - ✓ *~75mW for infrequent events*
  - ✓ *~50mW for frequent events.*

- ✓ *Latency significantly reduced by ESM*
- ✓ *Versus a pull model with a 16ms polling loop, ESM reduces latency by*
  - ✓ *~15ms for infrequent events*
  - ✓ *~12ms less latency for frequent events*
- ✓ *Versus a pull model with a 100ms polling loop, ESM reduces latency by*
  - ✓ *~95ms for infrequent events*
  - ✓ *~55ms for frequent events*



FUTURE DIRECTIONS

- Use the ESM to improve power consumption in other power hungry areas, such as 4G or WiFi
- Use the ESM to improve power consumption in adjacent areas, such as physical network layers
- Move display server from application layer to OS so that knowledge provided by ESM can be used to facilitate scheduling of apps
  - OS might shut down WiFi if foreground app does not require it
  - Drawing threads for background apps might be suspended
- Distribute event handling across cores: Use a lower-power consuming CPU core for OS event handling and higher power cores for application event handling.
- Examine middleware layers, such as .NET and Java to streamline their event propagation models.
- Measure the benefits of the ESM on cutting edge, 64-bit mobile CPUs, such as the NVIDIA TX1 system.

NVIDIA TX1. 64-bit, ARM Cortex A-57 (big.LITTLE architecture)

NVIDIA TK1. 32-bit, ARM Cortex A-15 (4+1 cores)

CONCLUSIONS

- Prototyping using the Android OS shows empirically that our model significantly reduces power consumption and latency when the device is somewhat or mostly idle.
- Power consumption and latency is still reduced when the device is receiving many events rapidly.
- Our ESM has room to grow and needs further research to apply it to other event-oriented systems that might needlessly consume power, such as Wifi or 4G (cellular).

LEARN MORE

Marz, S & Vander Zanden, B (2016). Reducing Power Consumption and Latency in Mobile Devices Using an Event Stream Model. ACM Trans. on Embed. Computing Syst., 16(1).

REFERENCES

1. Caroll, A., & Heiser, G. (2012). An Analysis of Power Consumption in a Smartphone. Paper presented at the USENIX Annual Technical Conference, Boston, MA, USA.
2. Cugola, G., Margara, A., Pezze,M., & Pradella, M. (2015). Efficient Analysis of Event Processing Applications. Paper presented at the Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, New York, NY, USA.
3. Frias-Martinez, V., & Virseda, J. (2012). On the relationship between socio-economic factors and cell phone usage. Paper presented at the Proceedings of the Fifth International Conference on Information and Communication Technologies and Development. Atlanta, GA, USA.
4. Han, S., Marshall, S., Chun, B.-G., & Ratnasamy, S. (2012). MegaPipe: A New Programming Interface for Scalable Network I/O. Paper presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), Hollywood, CA.
5. Lee, J., & Park, K. H. (2010). Interrupt handler migration and direct interrupt scheduling for rapid scheduling of interrupt-driven tasks. ACM Trans. Embed. Comput. Syst., 9(4), 1-34.