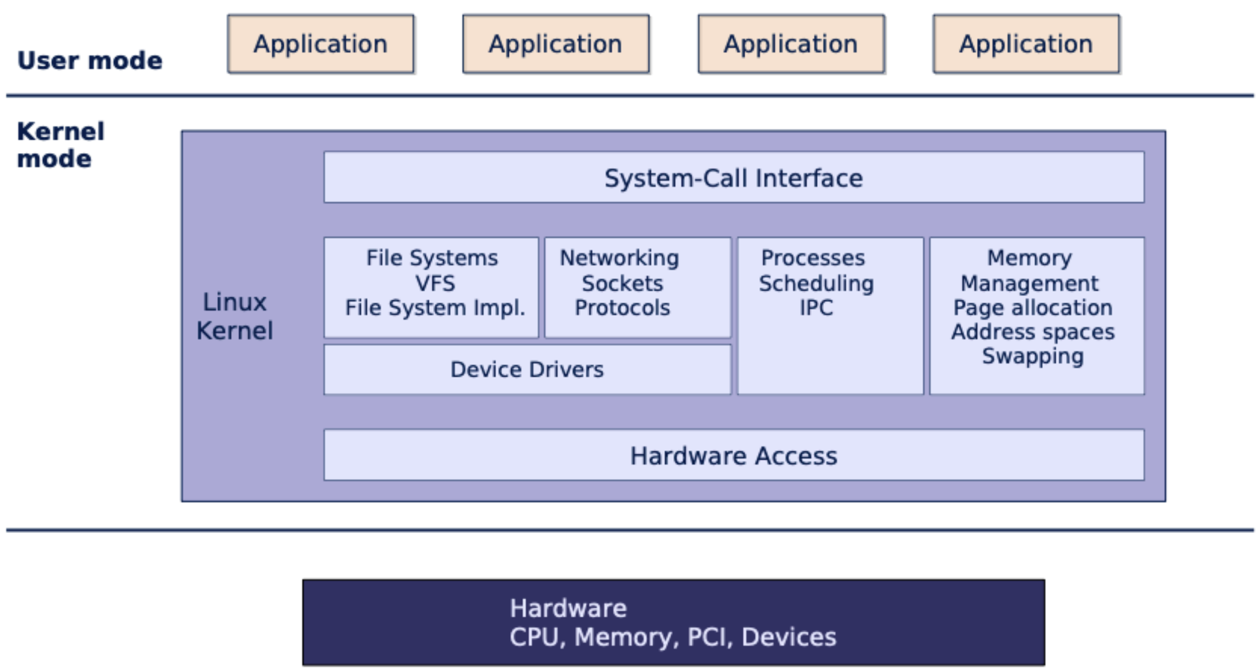
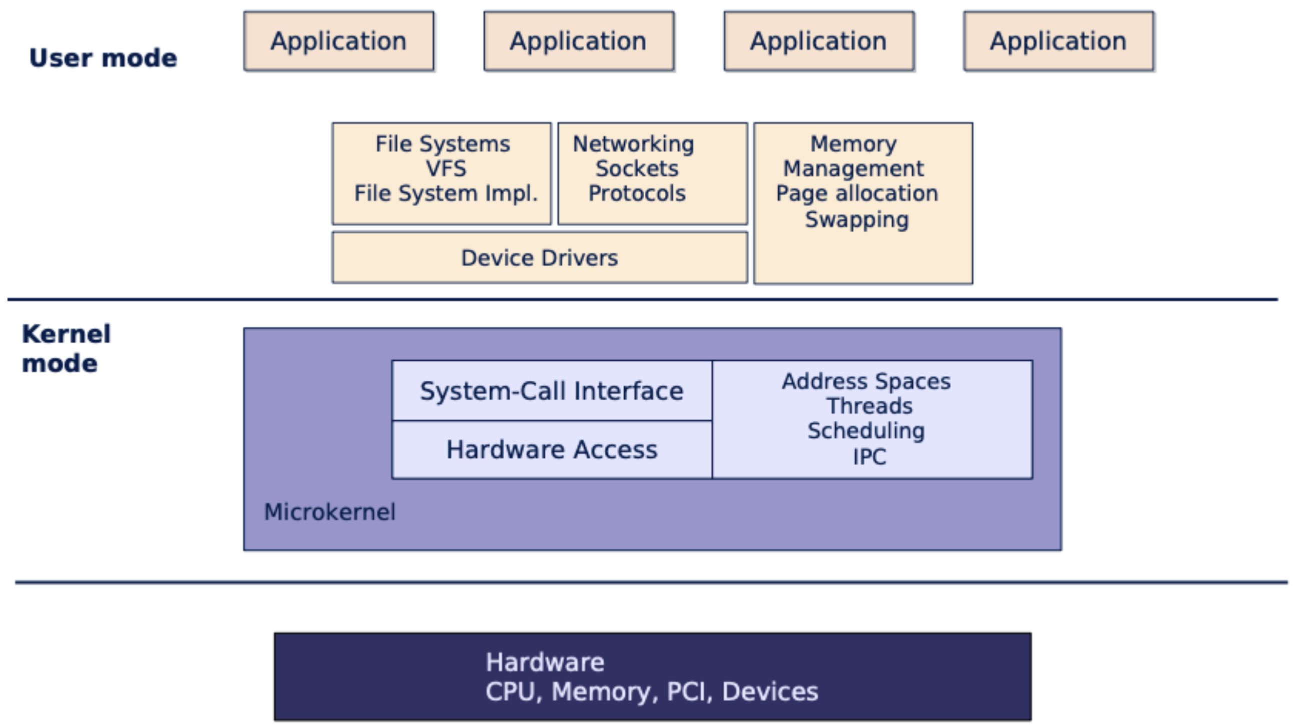
Scalable Operating System Design

Sebastian Ertel

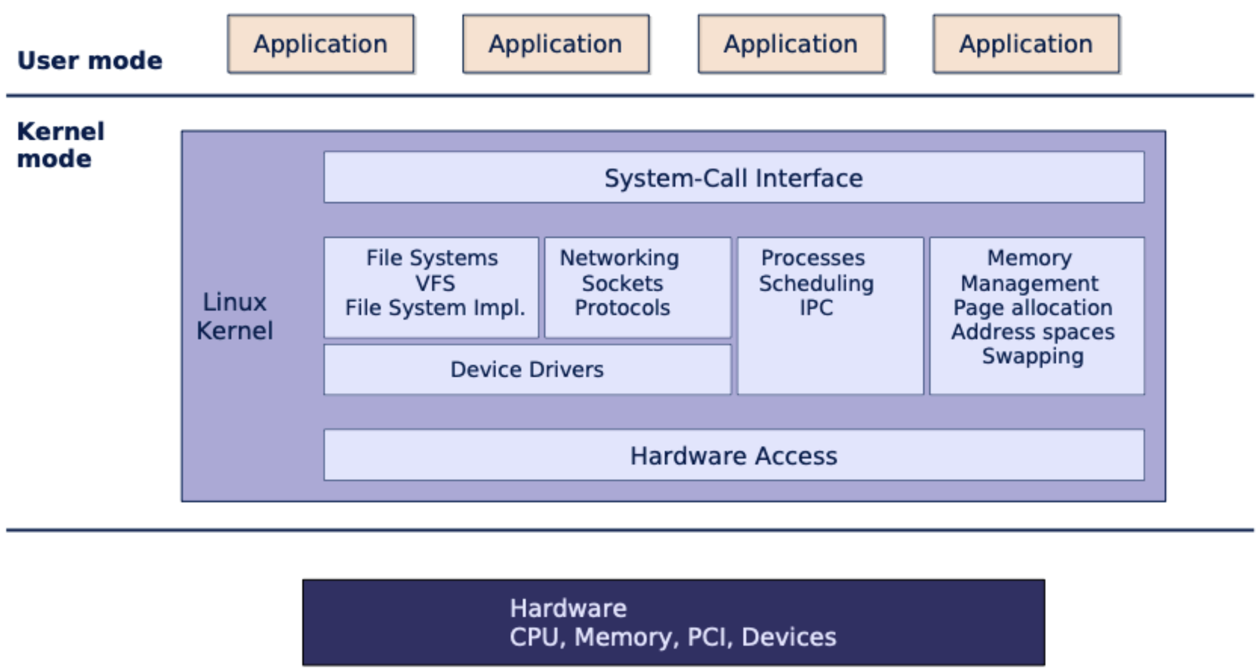
Monolithic vs Micro Kernel Design

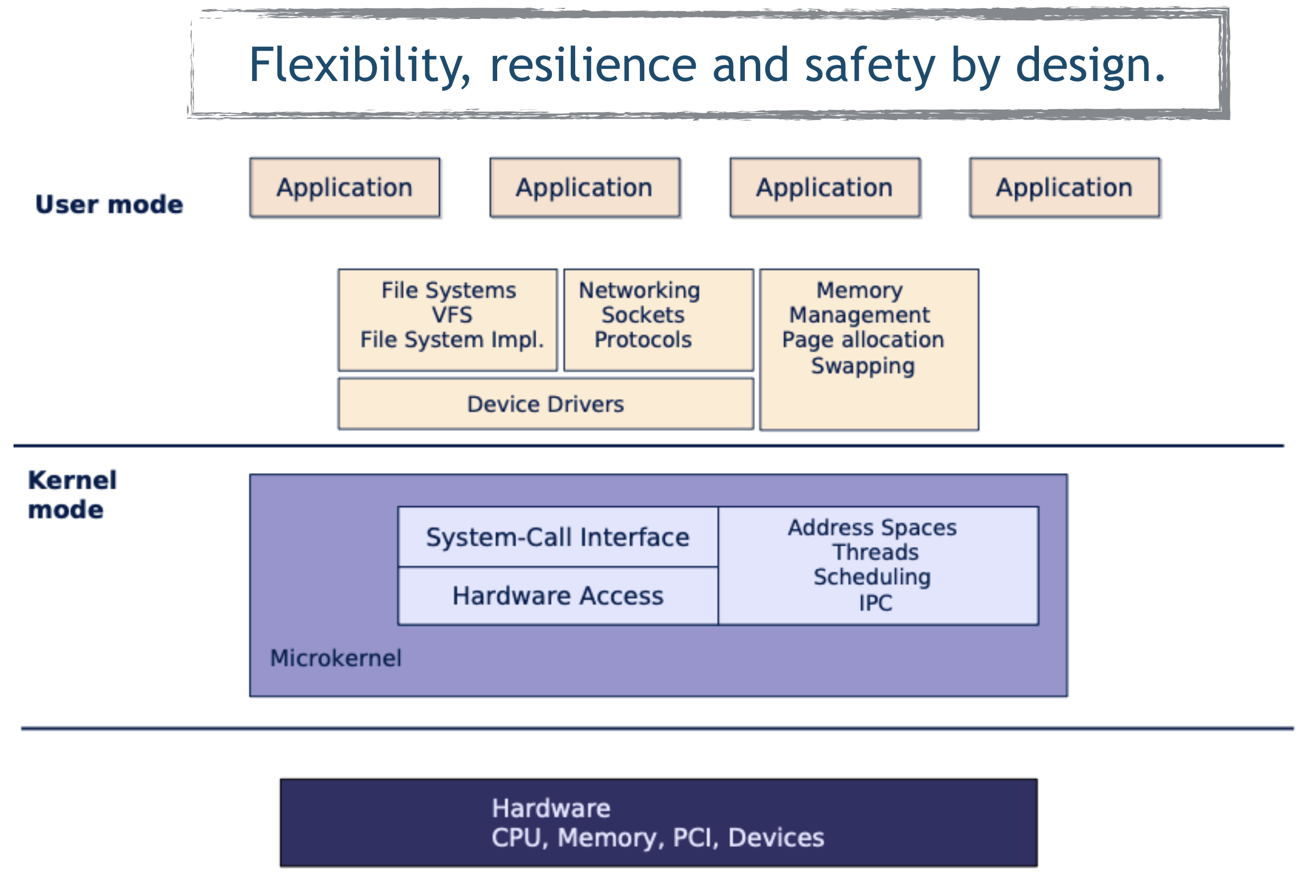




TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design

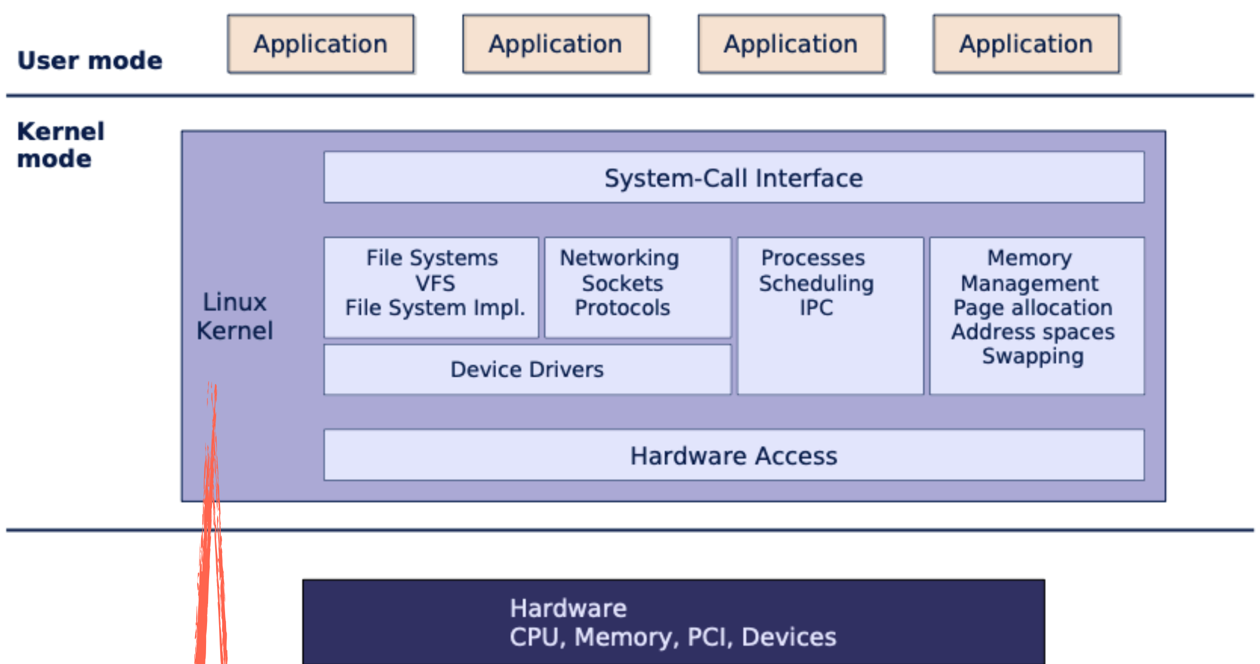


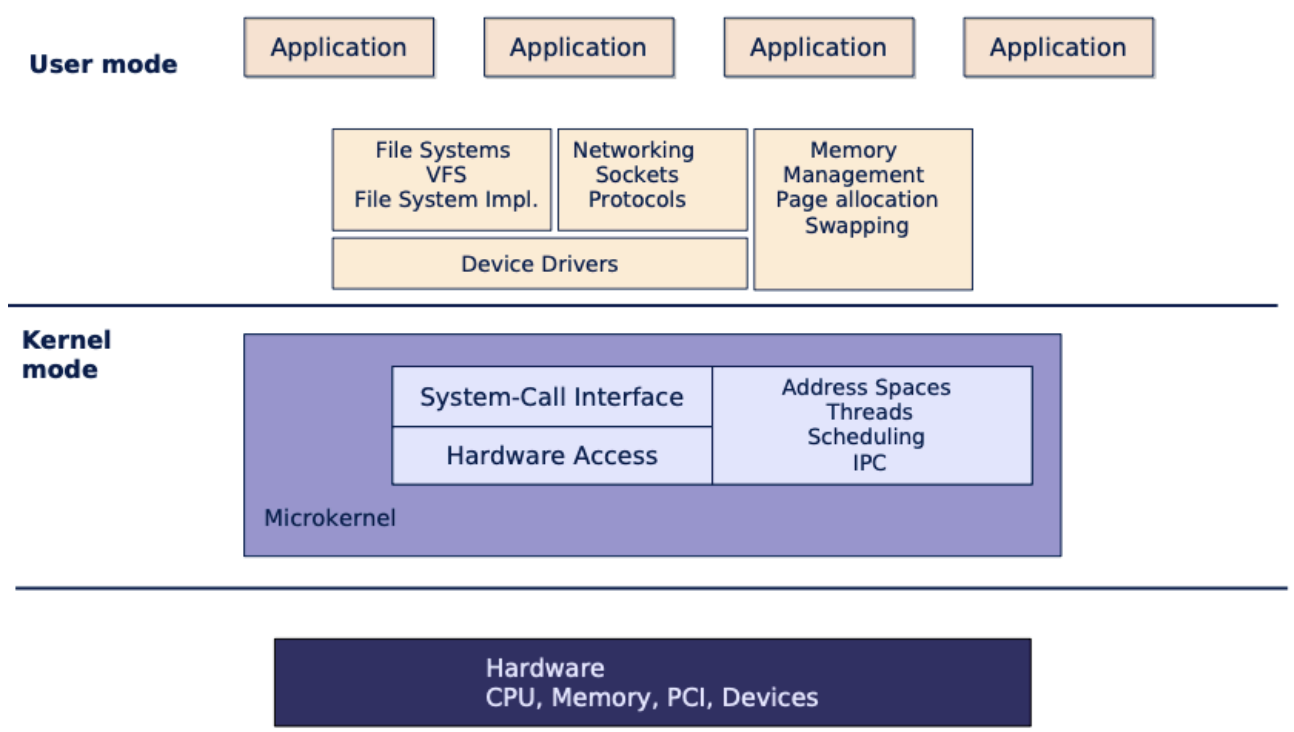


TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design

Flexibility, resilience and safety by design.

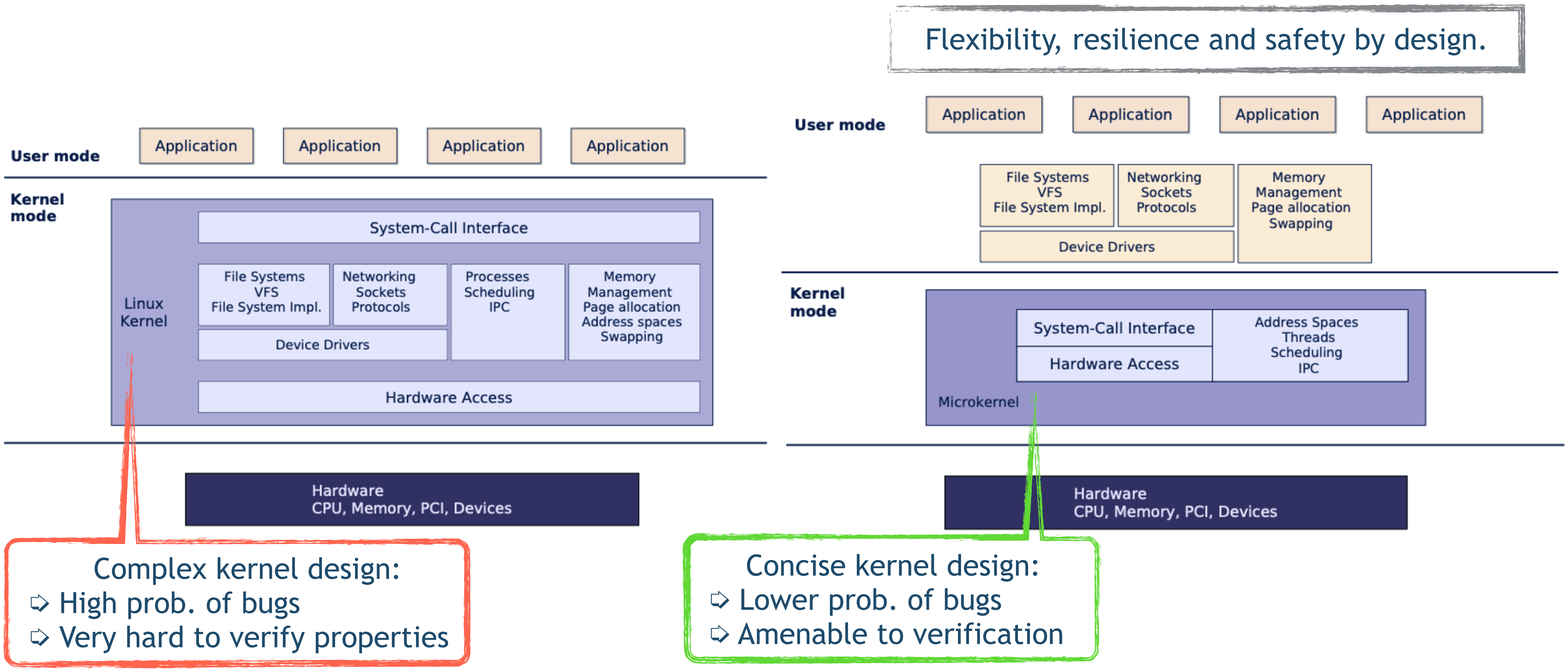




Complex kernel design:
- ➪ High prob. of bugs
- ➪ Very hard to verify properties

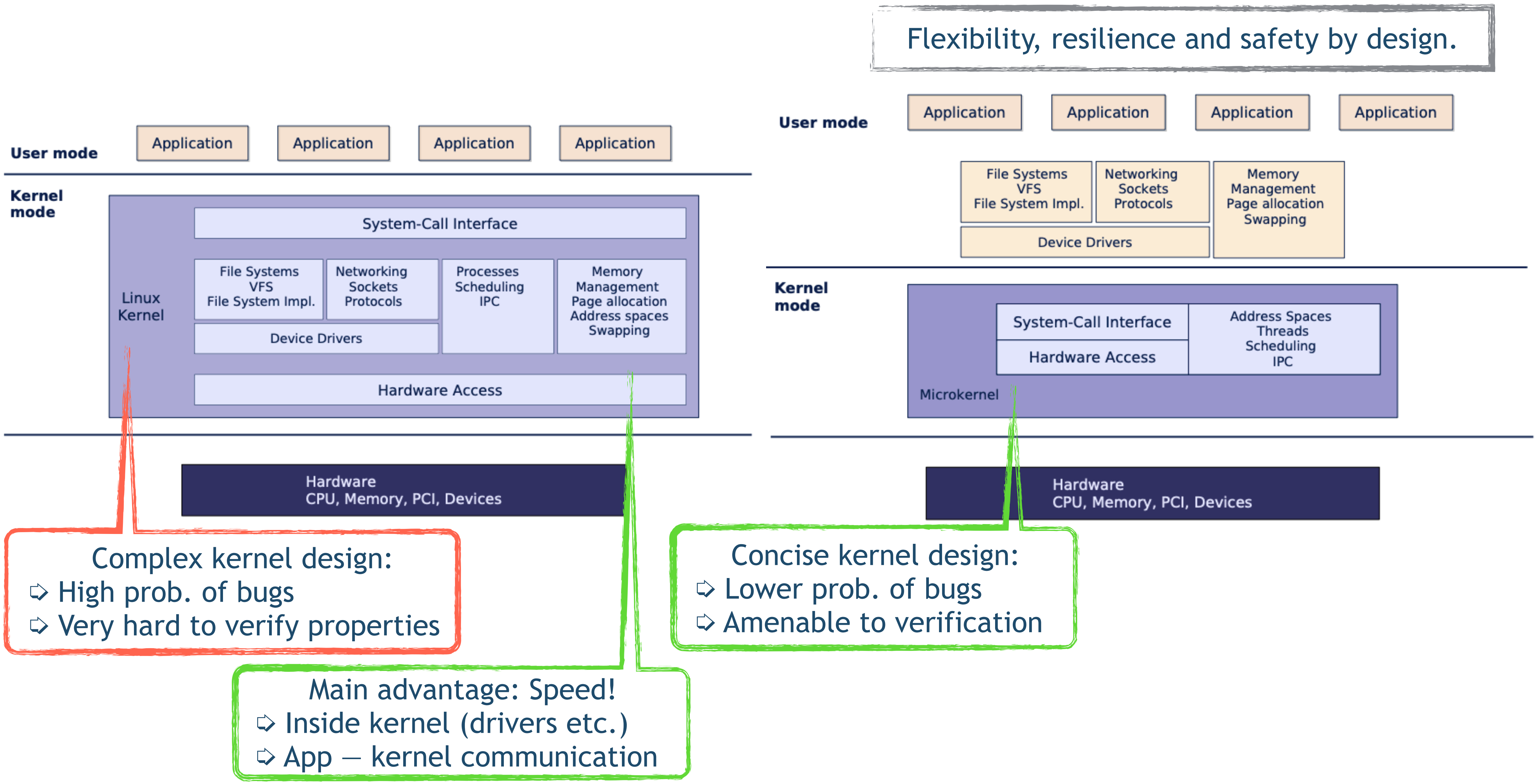
TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design



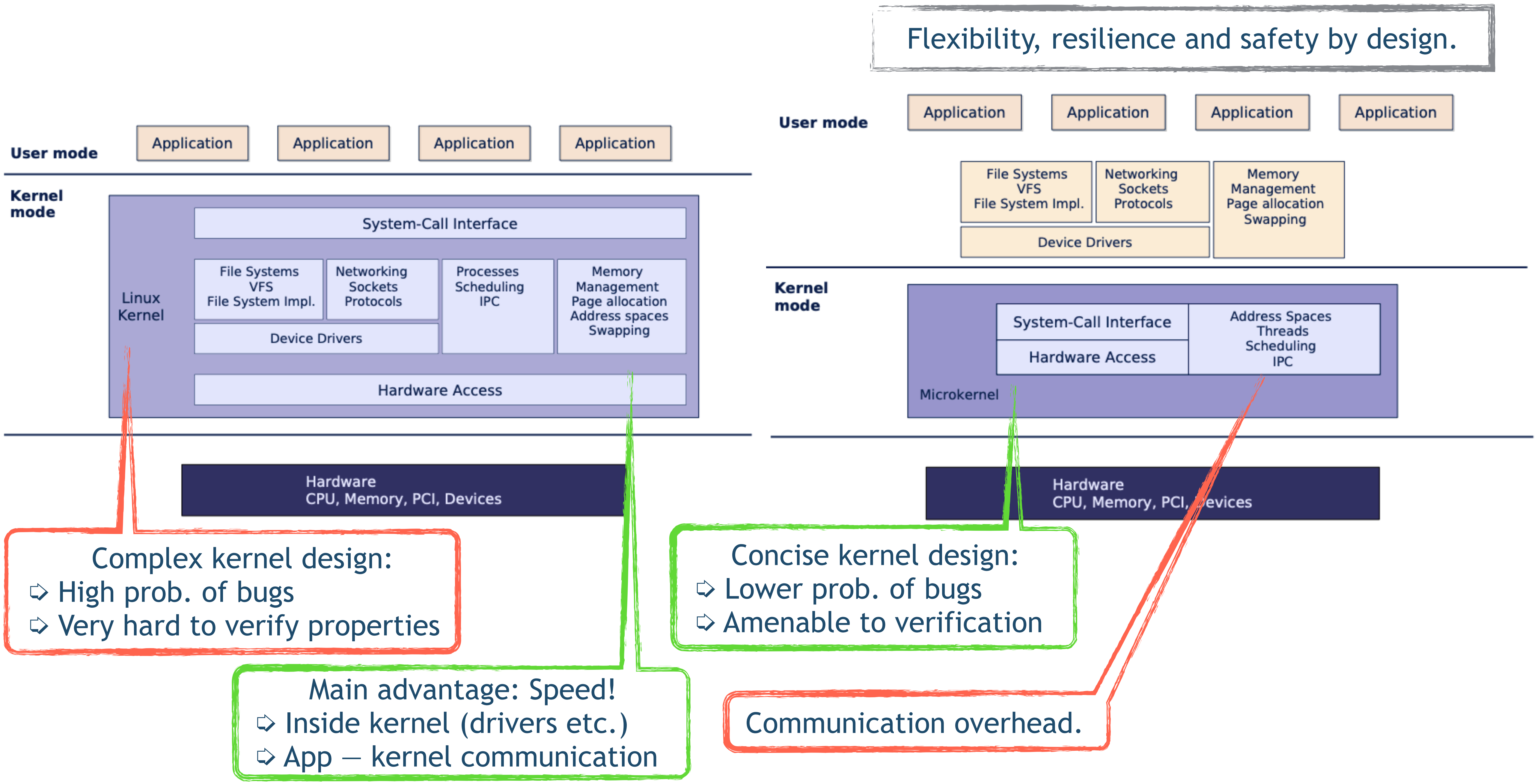
TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design



TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

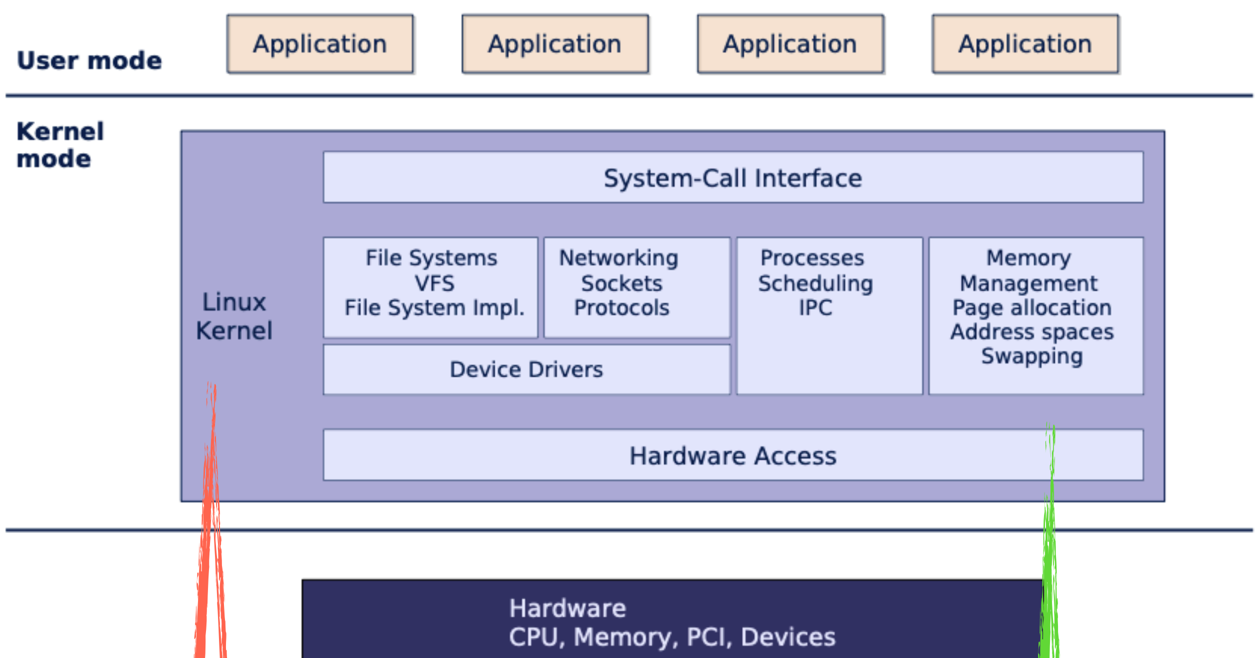
Monolithic vs Micro Kernel Design

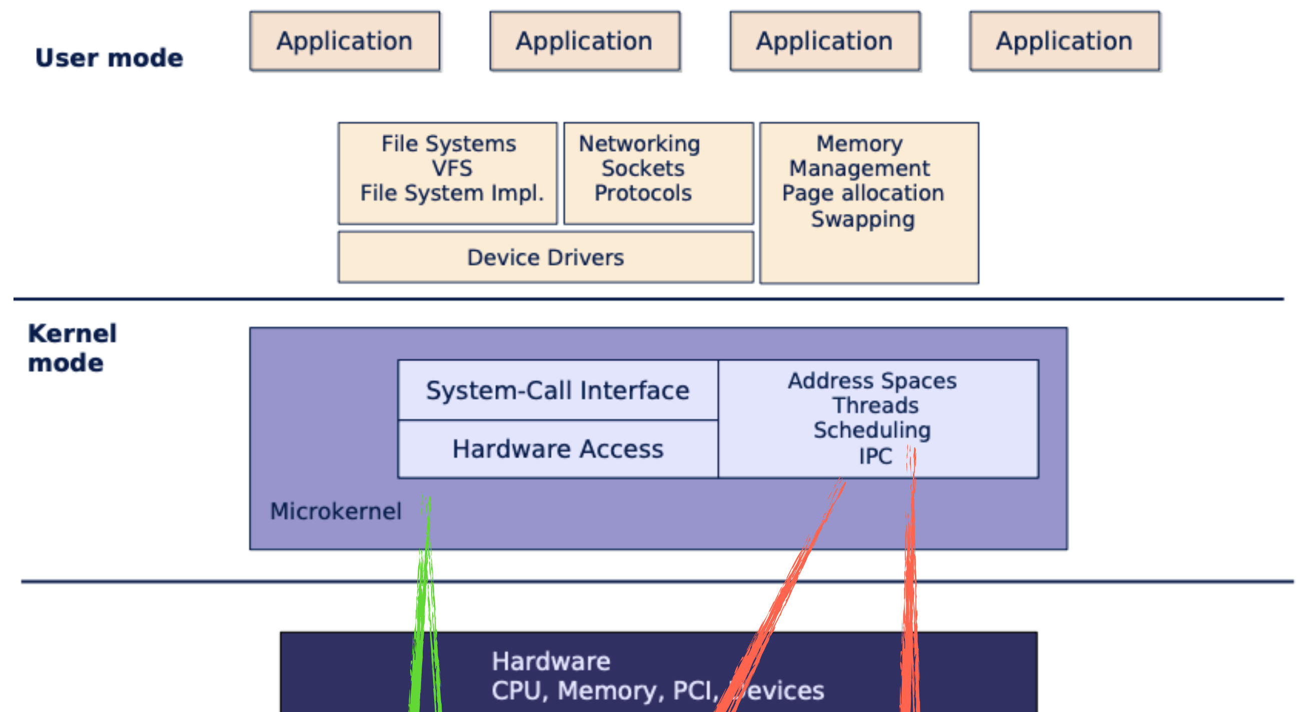


TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design

Flexibility, resilience and safety by design.





Complex kernel design:
- ➪ High prob. of bugs
- ➪ Very hard to verify properties

Main advantage: Speed!
- ➪ Inside kernel (drivers etc.)
- ➪ App – kernel communication

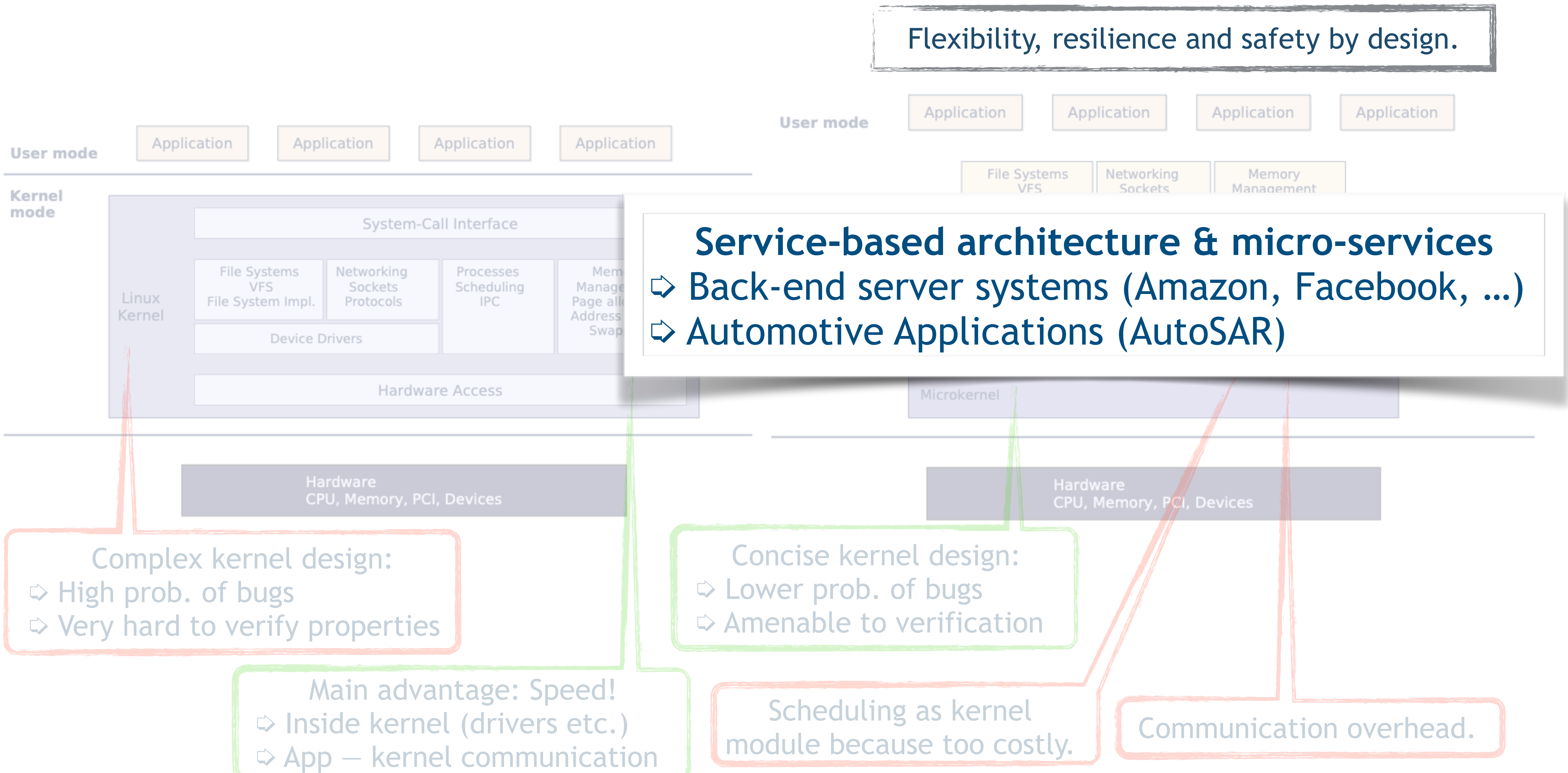
Concise kernel design:
- ➪ Lower prob. of bugs
- ➪ Amenable to verification

Communication overhead.

Scheduling as kernel module because too costly.

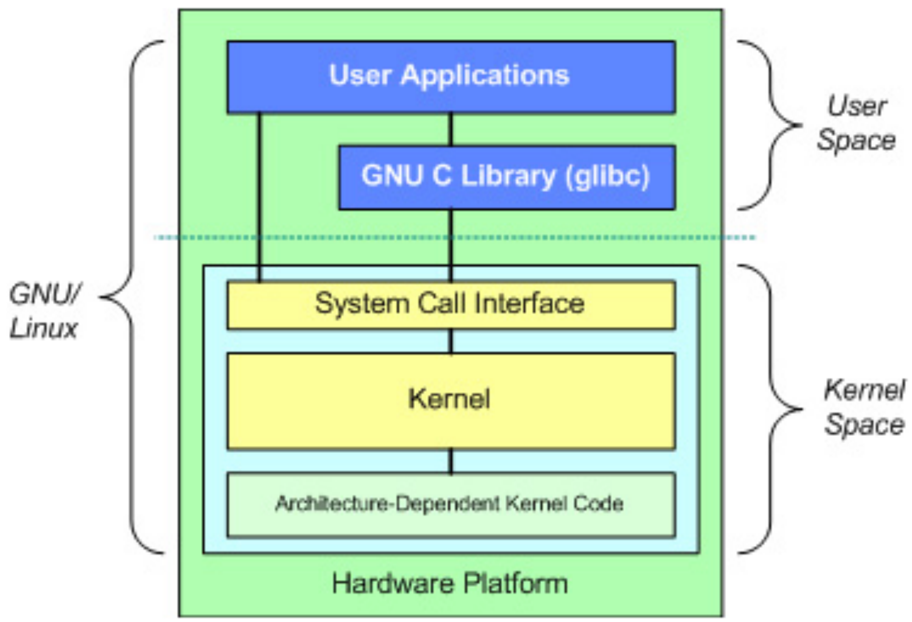
TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

Monolithic vs Micro Kernel Design



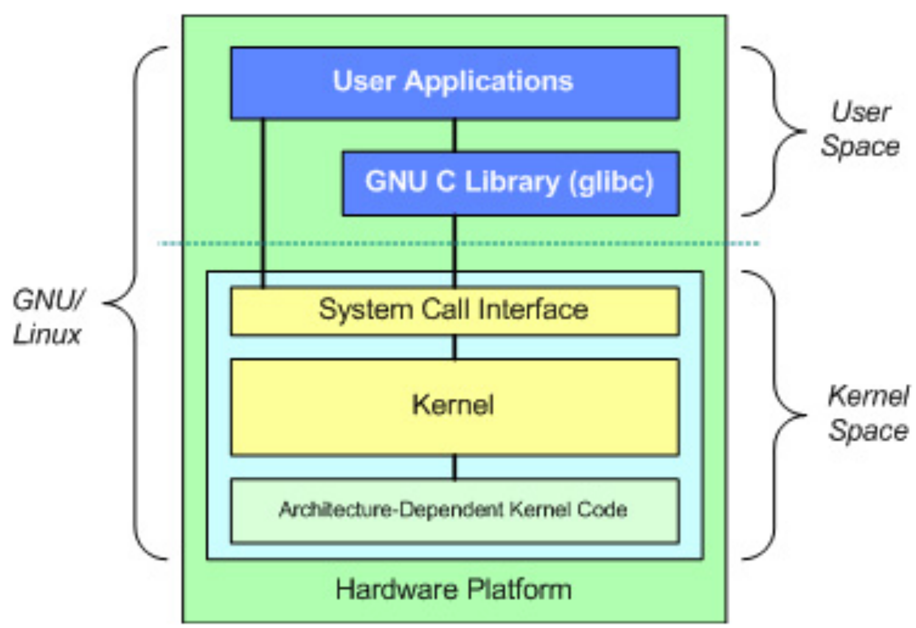
TU Dresden, Operating Systems Groups, Slides from Microkernel Lecture

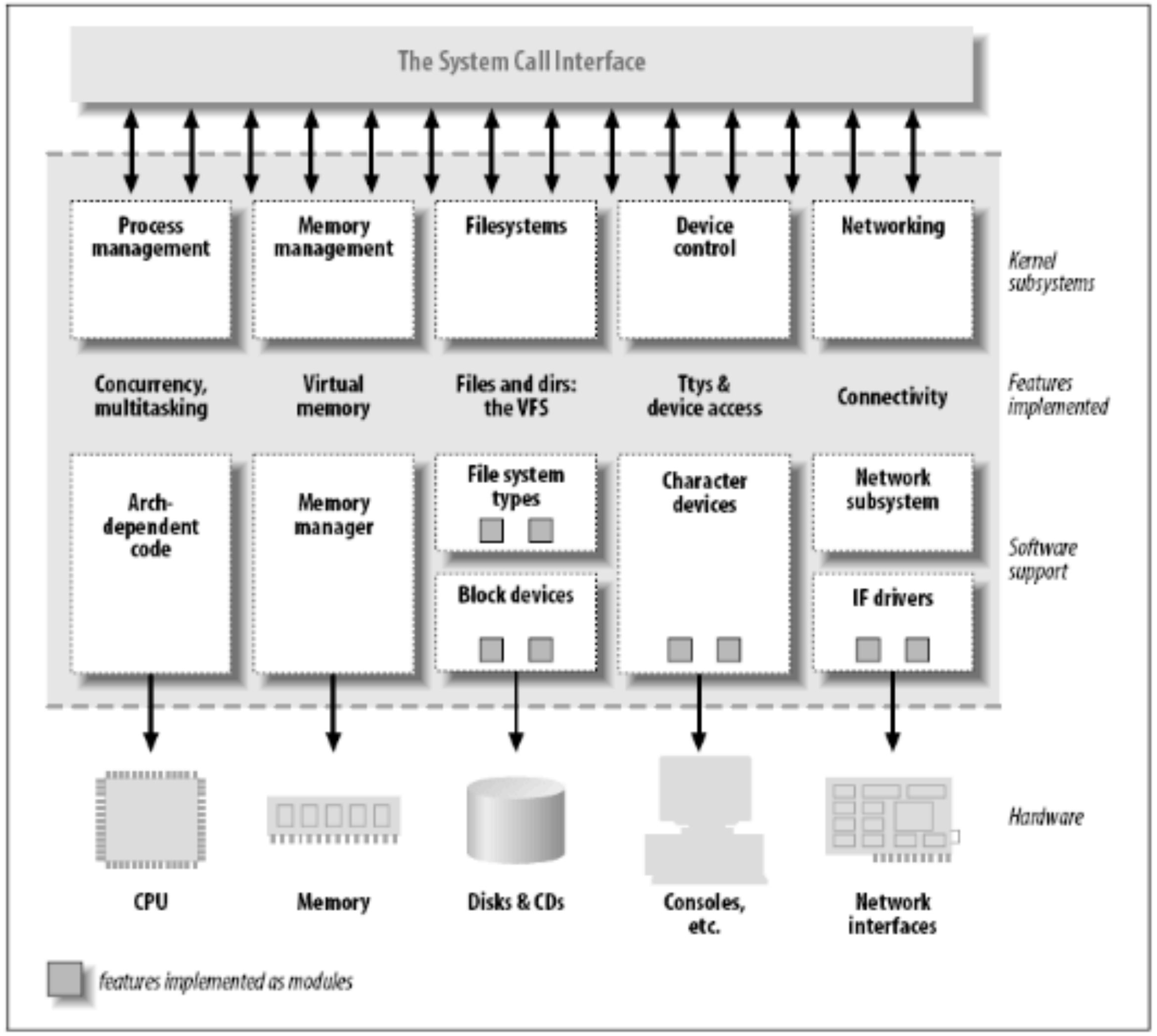
Linux Kernel



https://www.engineersgarage.com/tutorials/introduction-linux-part-715

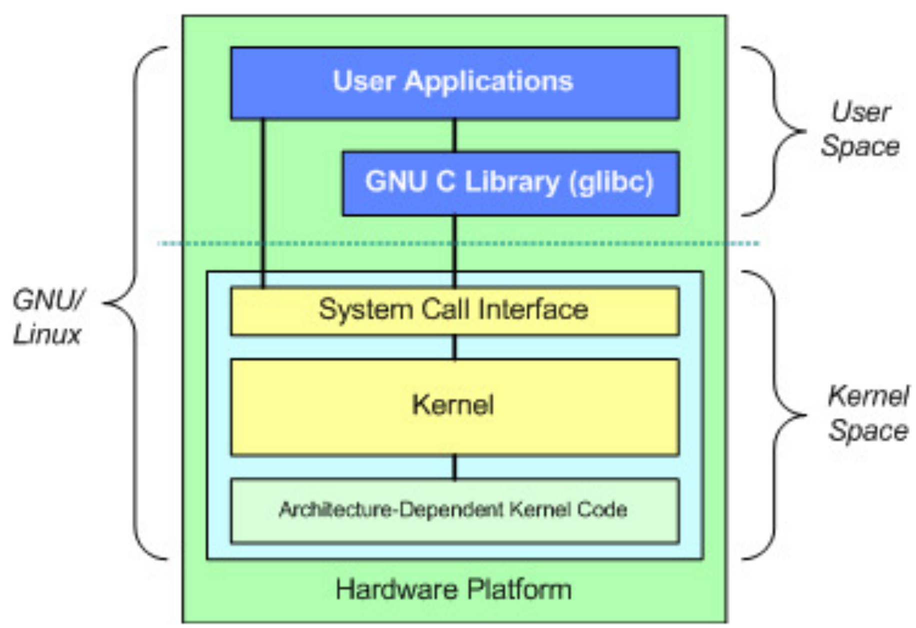
Linux Kernel

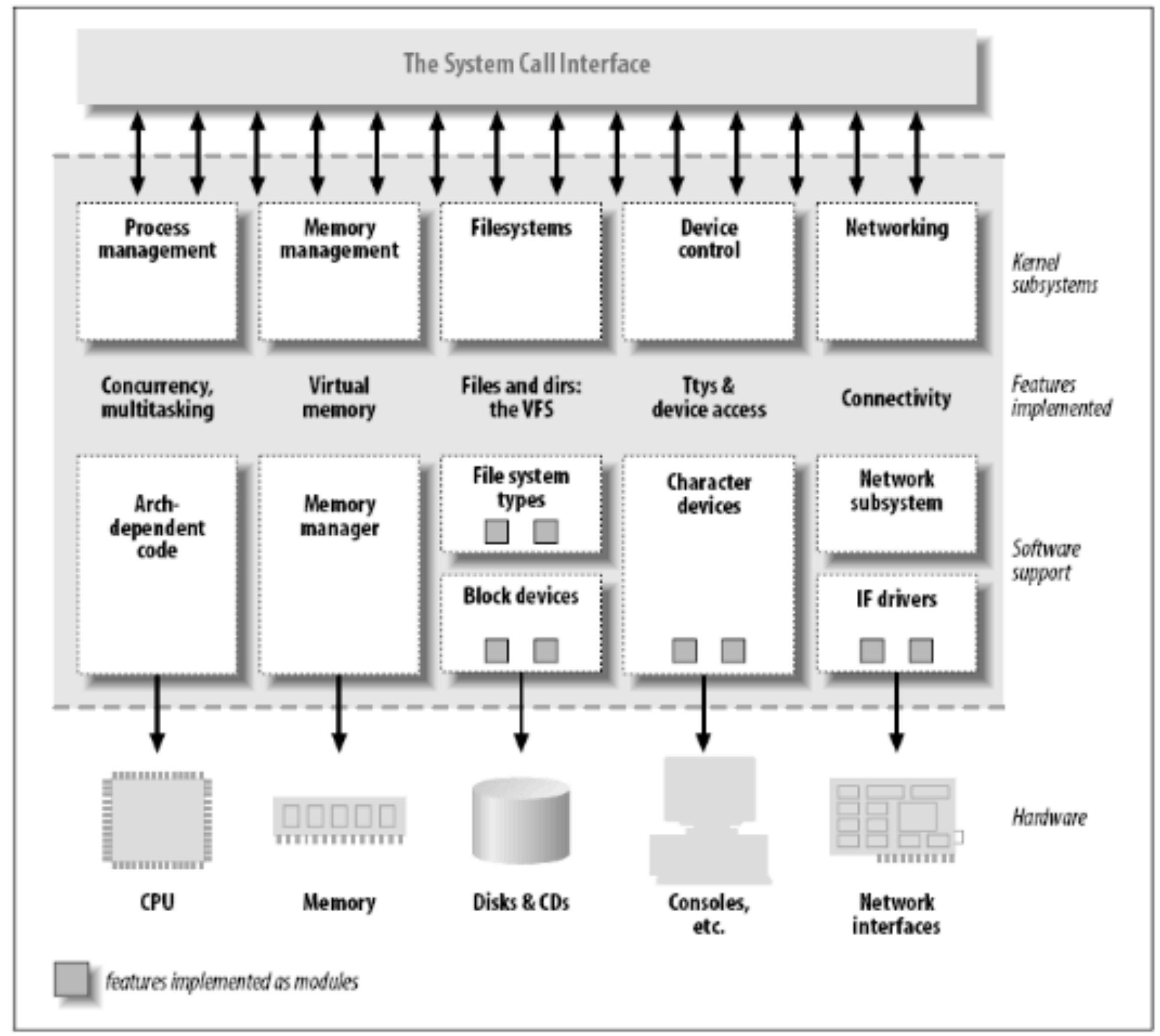




https://www.engineersgarage.com/tutorials/introduction-linux-part-715

Linux Kernel





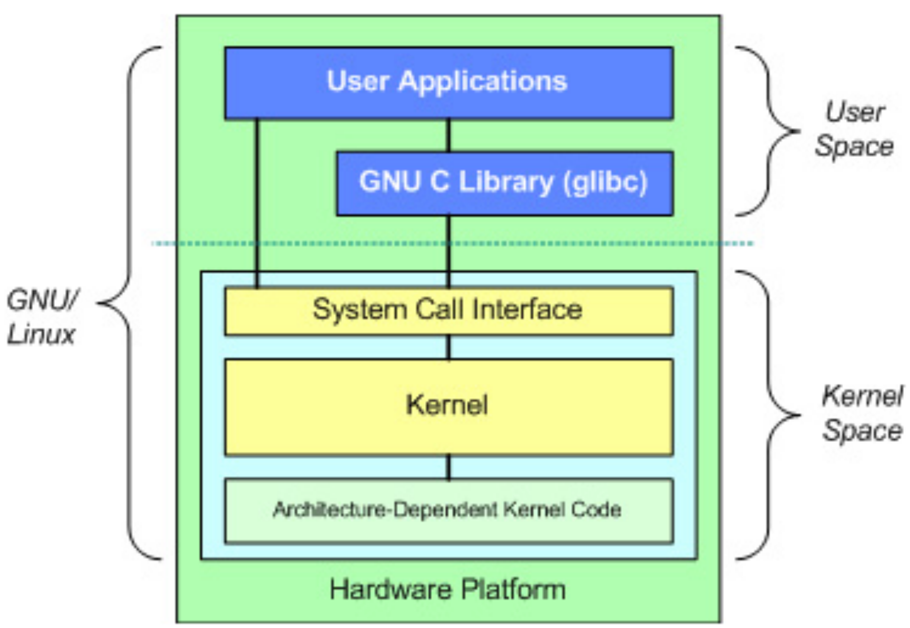
An OS kernel is inherently stateful!

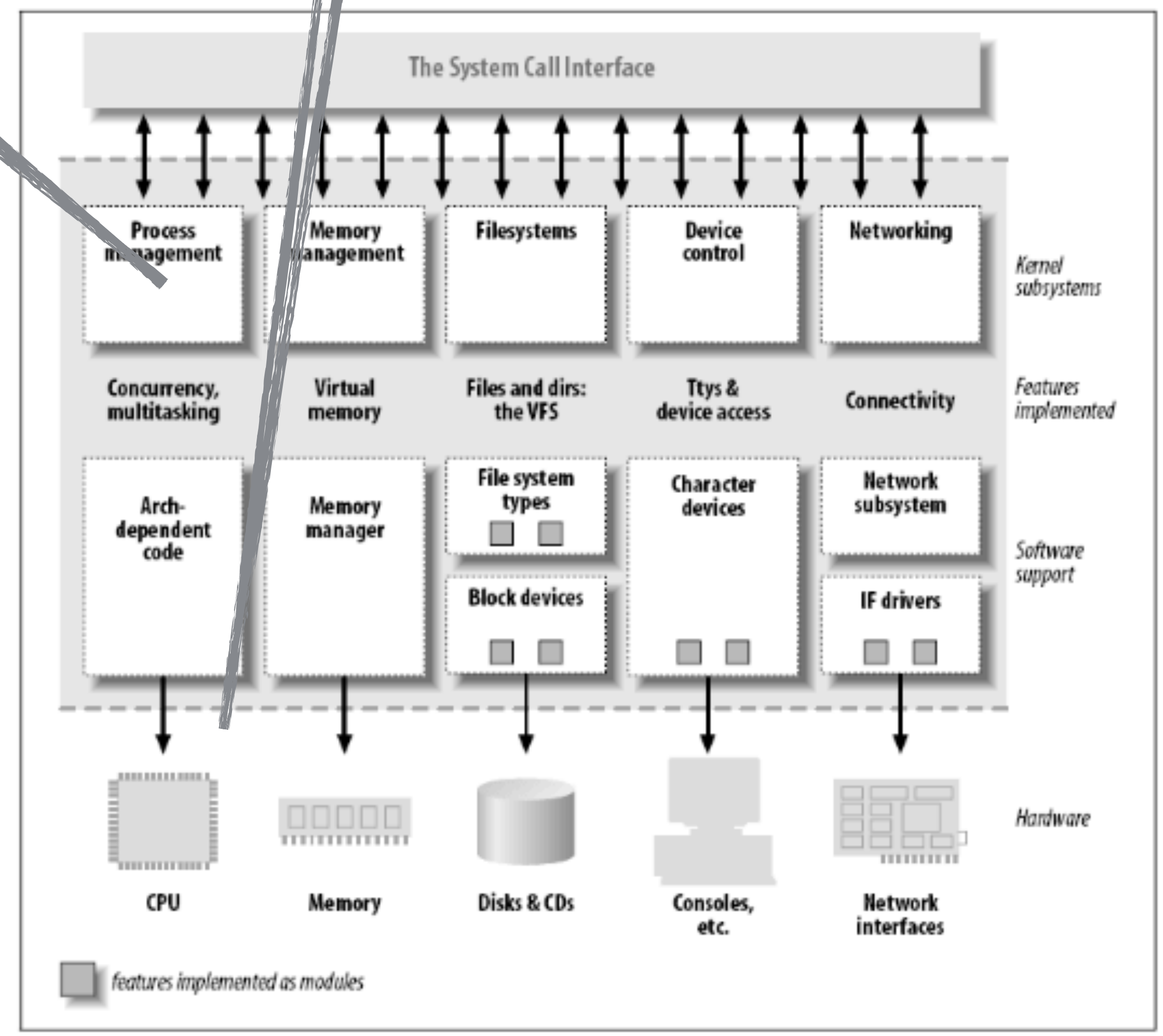
https://www.engineersgarage.com/tutorials/introduction-linux-part-715

Linux Kernel

Completely Fair Scheduler

Process Isolation





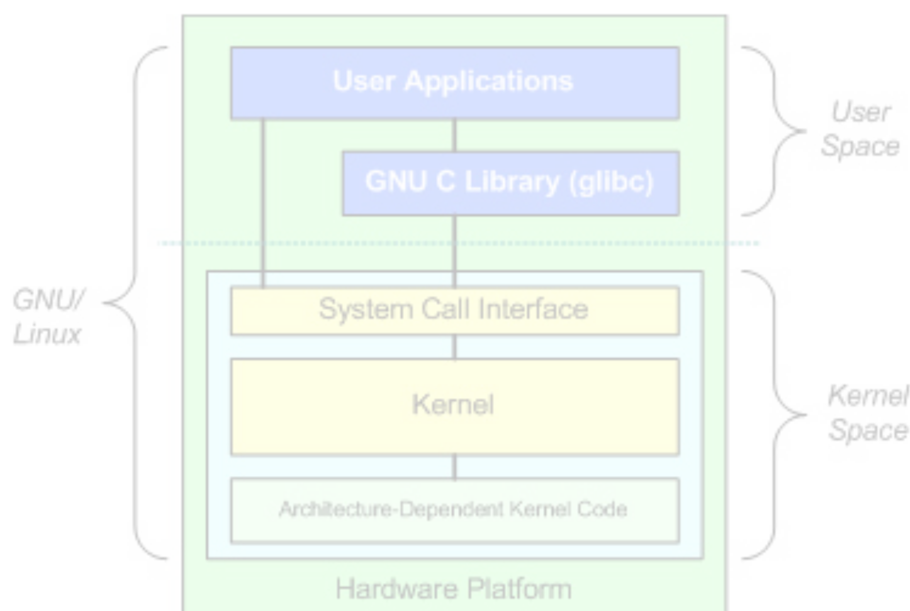
An OS kernel is inherently stateful!

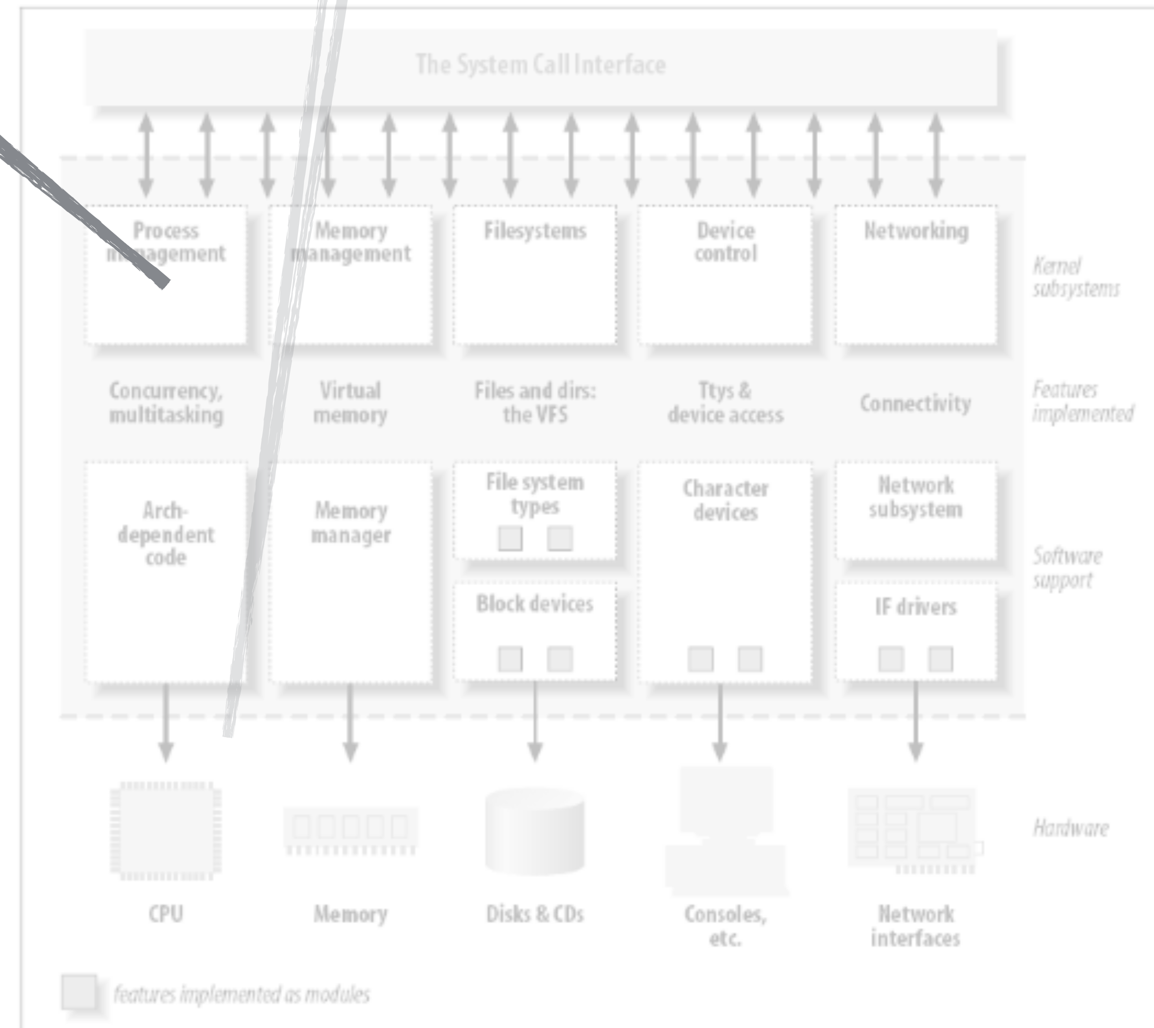
https://www.engineersgarage.com/tutorials/introduction-linux-part-715

Linux Kernel

Completely Fair Scheduler

Process Isolation



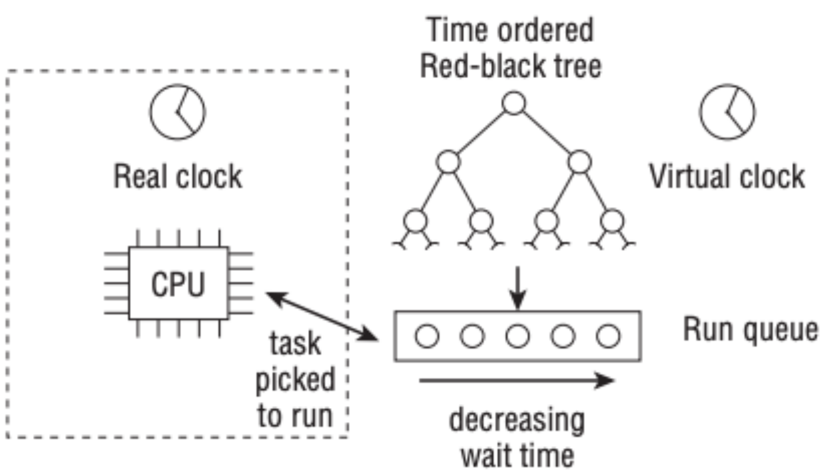


An OS kernel is inherently stateful!

---

https://www.engineersgarage.com/tutorials/introduction-linux-part-715

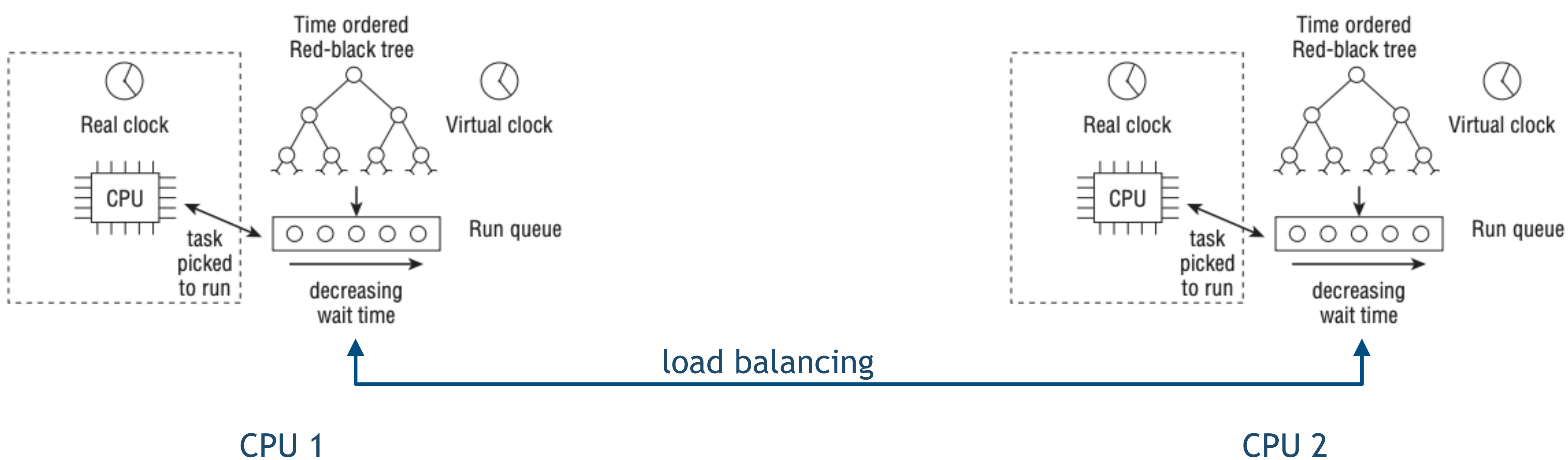
Linux Kernel
— Multi-core Architecture —



CPU 1

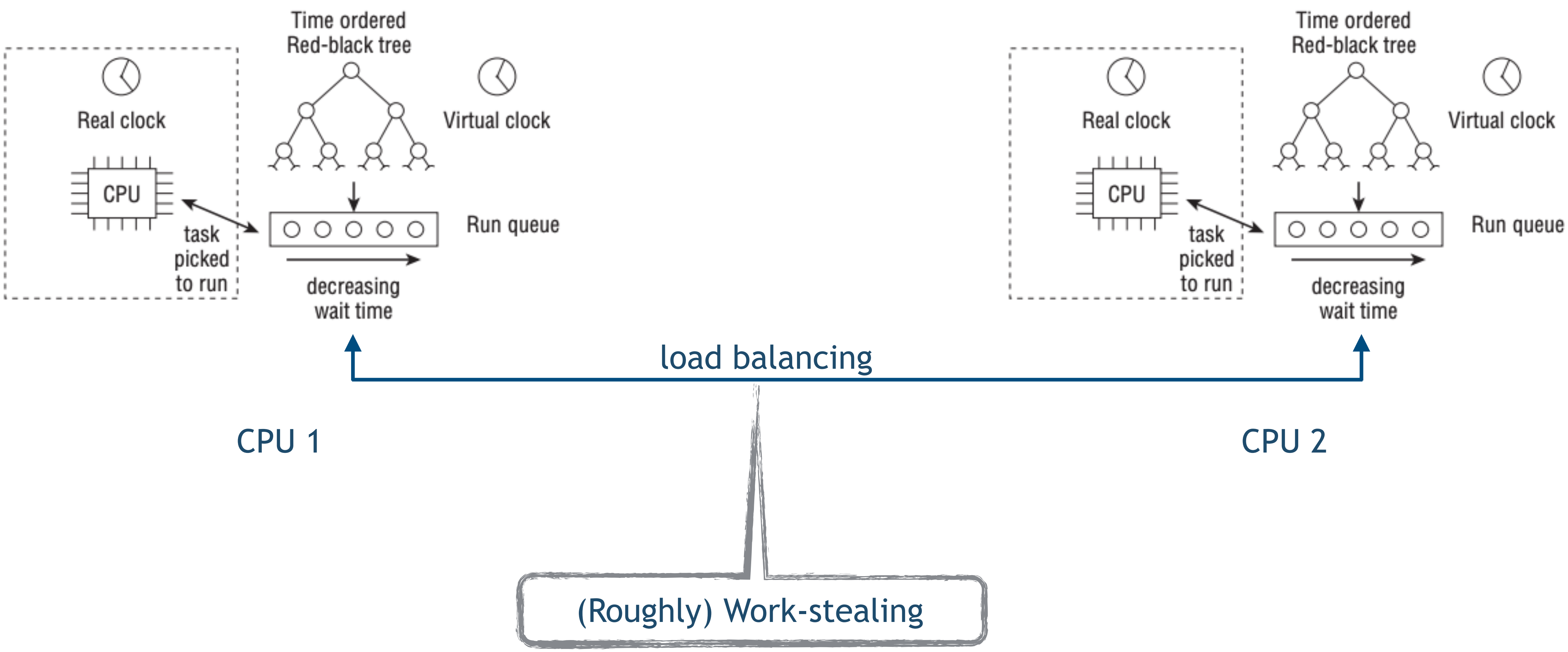
Mauerer, W., 2010. *Professional Linux kernel architecture*. John Wiley & Sons.

Linux Kernel
— Multi-core Architecture —



Mauerer, W., 2010. *Professional Linux kernel architecture*. John Wiley & Sons.

Linux Kernel
— Multi-core Architecture —



Mauerer, W., 2010. *Professional Linux kernel architecture*. John Wiley & Sons.

Multikernel — Barrelfish

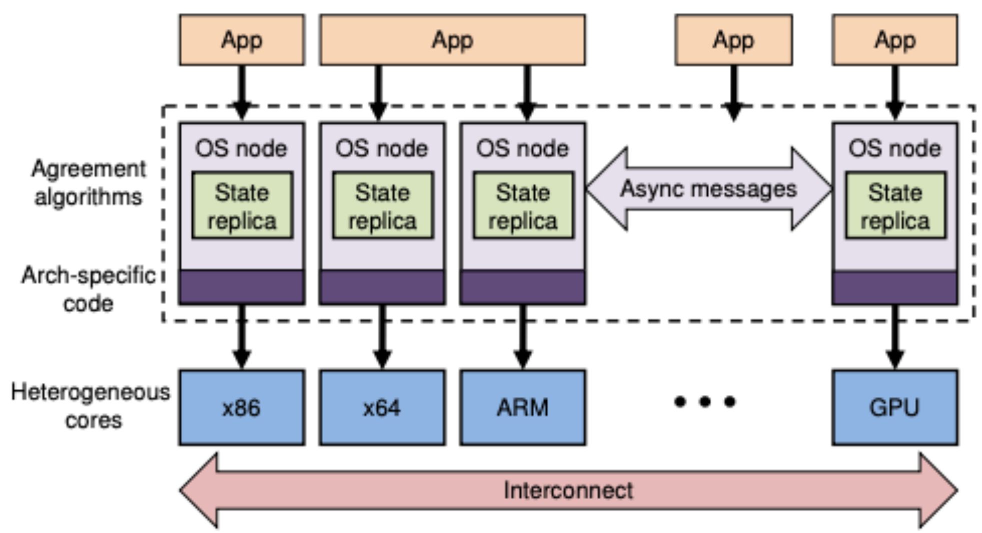


Figure 1: The multikernel model.

Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A., 2009, October. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 29-44). ACM.

Multikernel — Barrelfish

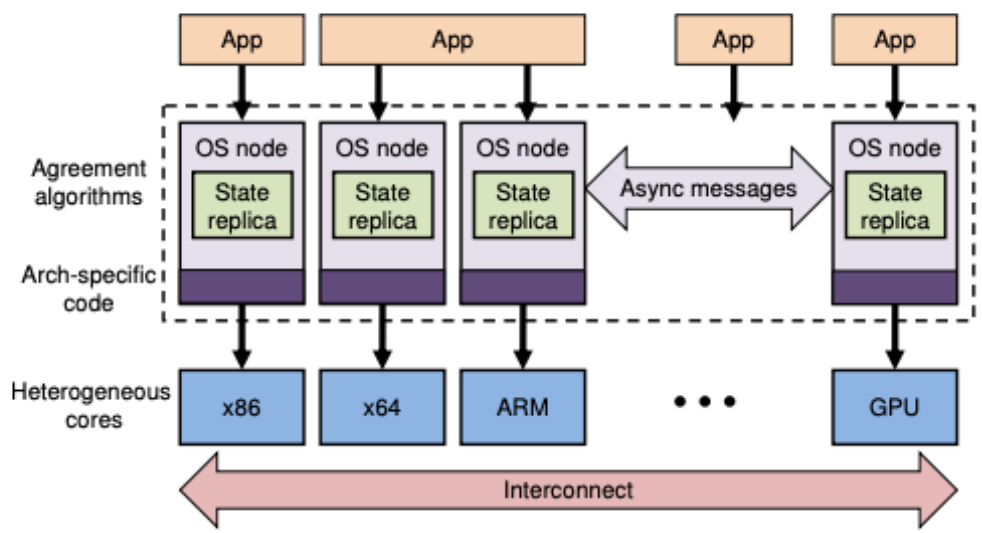


Figure 1: The multikernel model.

Highlights:

- no shared memory (state) (/HW cache coherency)
- HW message passing (batching)
- replicated state with 2-phase commit coordination

Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A., 2009, October. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 29-44). ACM.

Multikernel — Barrelfish

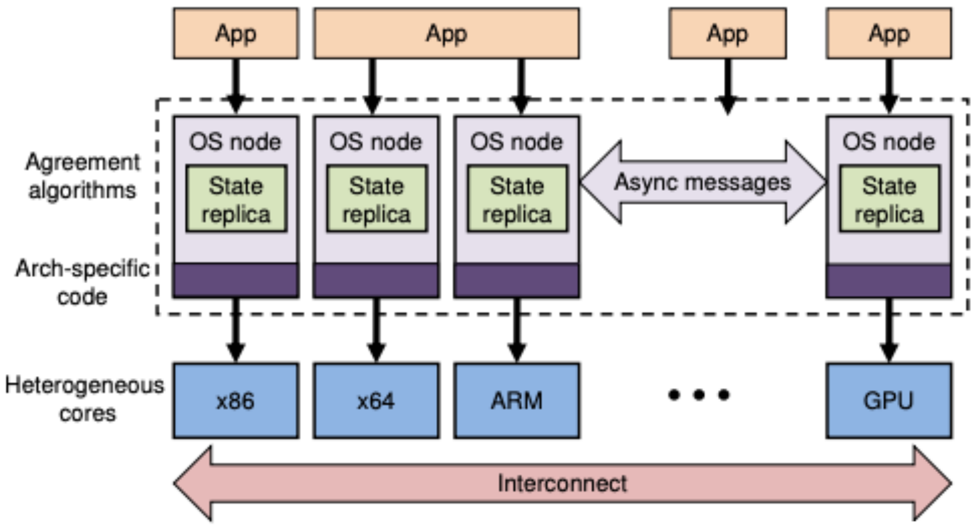


Figure 1: The multikernel model.

Highlights:

- no shared memory (state) (/HW cache coherency)
- HW message passing (batching)
- replicated state with 2-phase commit coordination

Figure 4: Spectrum of sharing and locking disciplines.

Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A., 2009, October. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 29-44). ACM.

Multikernel — Barrelfish

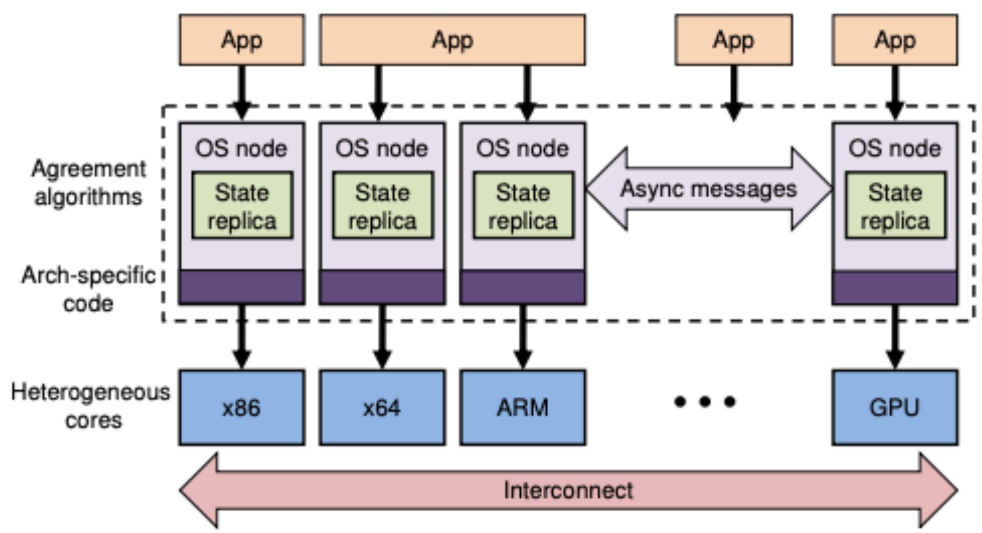


Figure 1: The multikernel model.

Highlights:

- no shared memory (state) (/HW cache coherency)
- HW message passing (batching)
- replicated state with 2-phase commit coordination

Figure 4: Spectrum of sharing and locking disciplines.

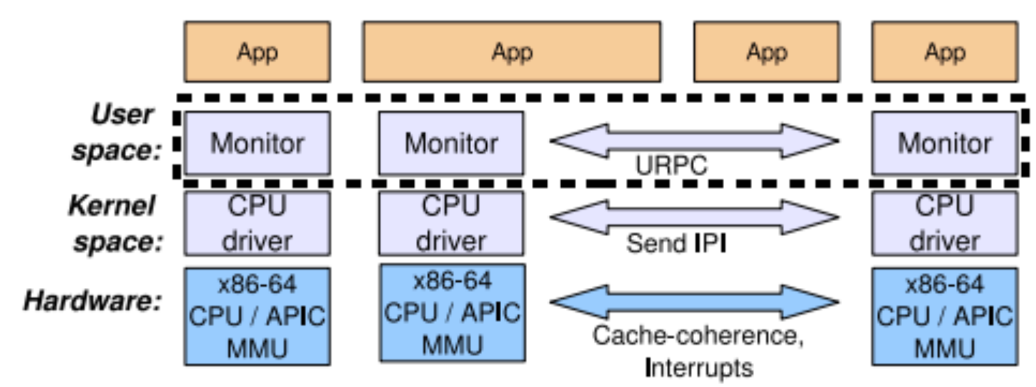


Figure 5: Barrelfish structure

Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A., 2009, October. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 29-44). ACM.

Multikernel — Barrelfish

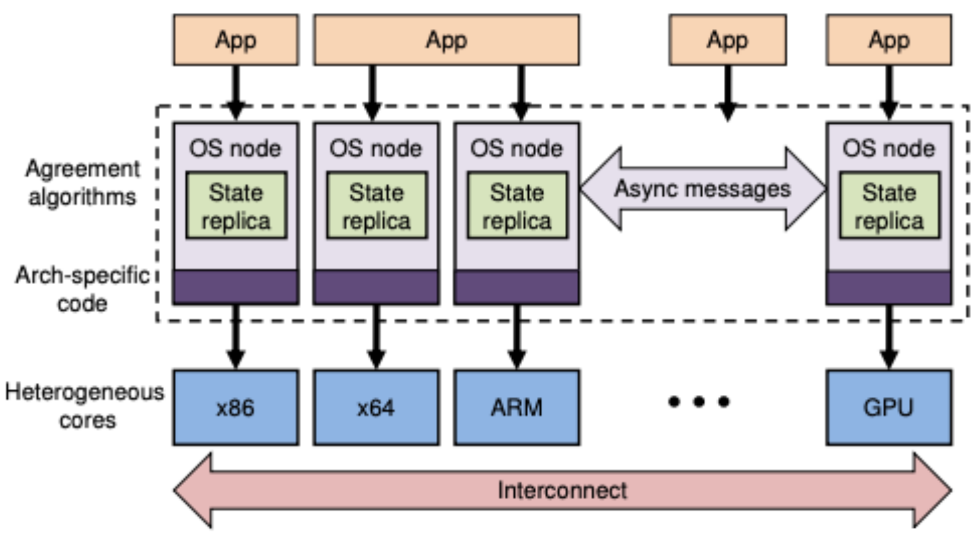


Figure 1: The multikernel model.

Highlights:

- no shared memory (state) (/HW cache coherency)
- HW message passing (batching)
- replicated state with 2-phase commit coordination

Figure 4: Spectrum of sharing and locking disciplines.

Analysis:

- great modular design for heterogeinity
- replicated state == replicated cache
- 2-phase commit vs. HW cache coherency

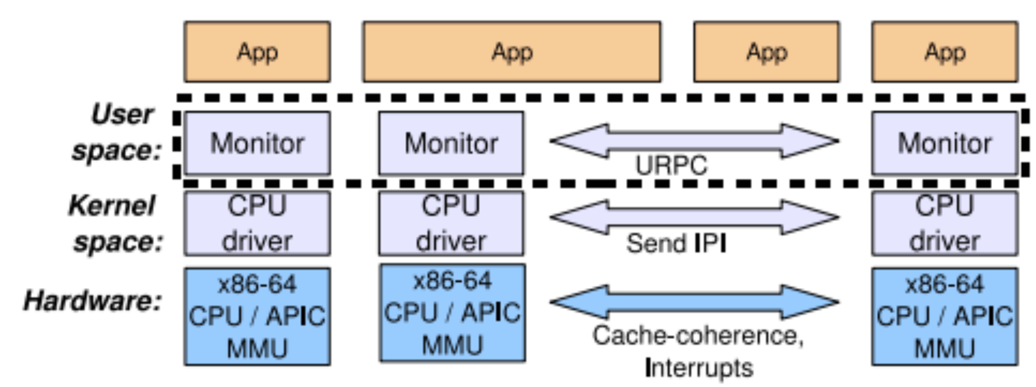


Figure 5: Barrelfish structure

Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A., 2009, October. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 29-44). ACM.

Linux on Multi-/Many-core

Monolithic kernels can scale:

- "Just" pick the right locks!
- Make side-effects local whenever possible!
- Use "new" optimizations (sloppy counters)!

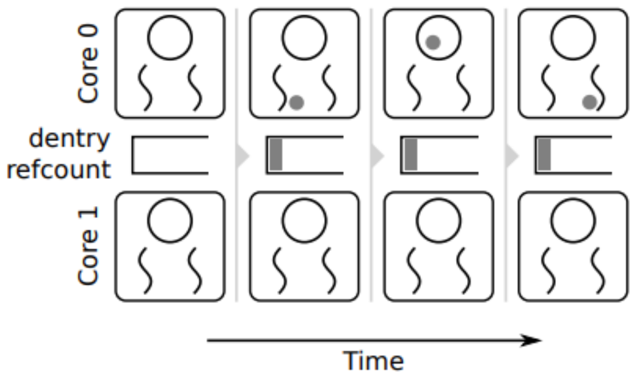
Boyd-Wickizer, S., Clements, A.T., Mao, Y., Pesterev, A., Kaashoek, M.F., Morris, R.T. and Zeldovich, N., 2010, October. An Analysis of Linux Scalability to Many Cores. In *OSDI* (Vol. 10, No. 13, pp. 86-93).

Lozi, J.P., Lepers, B., Funston, J., Gaud, F., Quéma, V. and Fedorova, A., 2016, April. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems* (p. 1). ACM.

Linux on Multi-/Many-core

Monolithic kernels can scale:

- "Just" pick the right locks!
- Make side-effects local whenever possible!
- Use "new" optimizations (sloppy counters)!



Boyd-Wickizer, S., Clements, A.T., Mao, Y., Pesterev, A., Kaashoek, M.F., Morris, R.T. and Zeldovich, N., 2010, October. An Analysis of Linux Scalability to Many Cores. In *OSDI* (Vol. 10, No. 13, pp. 86-93).

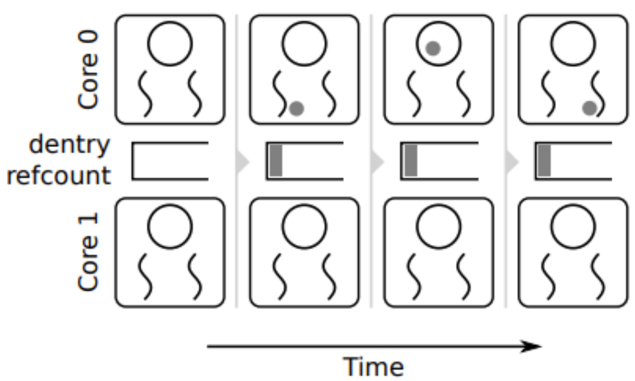
Lozi, J.P., Lepers, B., Funston, J., Gaud, F., Quéma, V. and Fedorova, A., 2016, April. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems* (p. 1). ACM.

Linux on Multi-/Many-core

Monolithic kernels can scale:

- “Just” pick the right locks!
- Make side-effects local whenever possible!
- Use “new” optimizations (sloppy counters)!

> Performance is often the enemy of scaling. One way to achieve scalability is to use inefficient algorithms, so that each core busily computes and makes little use of shared resources such as locks. Conversely, increasing the efficiency of software often makes it less scalable, by increasing the fraction of time it uses shared resources.



Boyd-Wickizer, S., Clements, A.T., Mao, Y., Pesterev, A., Kaashoek, M.F., Morris, R.T. and Zeldovich, N., 2010, October. An Analysis of Linux Scalability to Many Cores. In *OSDI* (Vol. 10, No. 13, pp. 86-93).

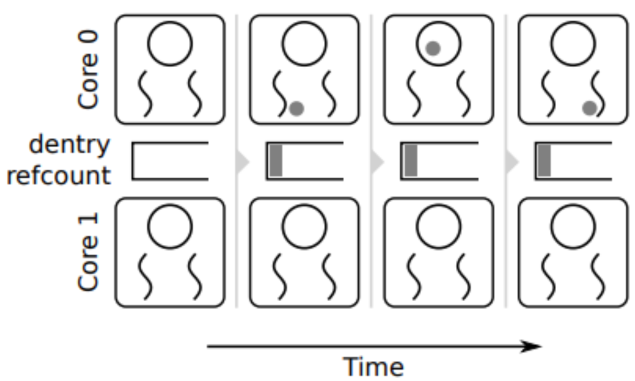
Lozi, J.P., Lepers, B., Funston, J., Gaud, F., Quéma, V. and Fedorova, A., 2016, April. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems* (p. 1). ACM.

Linux on Multi-/Many-core

Monolithic kernels can scale:

- "Just" pick the right locks!
- Make side-effects local whenever possible!
- Use "new" optimizations (sloppy counters)!

> Performance is often the enemy of scaling. One way to achieve scalability is to use inefficient algorithms, so that each core busily computes and makes little use of shared resources such as locks. Conversely, increasing the efficiency of software often makes it less scalable, by increasing the fraction of time it uses shared resources.



Architecture-dependent optimizations:

- often spread across various routines in the kernel/scheduler and
- result in semantic composability problems of functions.

Boyd-Wickizer, S., Clements, A.T., Mao, Y., Pesterev, A., Kaashoek, M.F., Morris, R.T. and Zeldovich, N., 2010, October. An Analysis of Linux Scalability to Many Cores. In *OSDI* (Vol. 10, No. 13, pp. 86-93).

Lozi, J.P., Lepers, B., Funston, J., Gaud, F., Quéma, V. and Fedorova, A., 2016, April. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems* (p. 1). ACM.

Take-away Points

Support for heterogeneity:

- Good separation of architectural aspects results in simpler and more concise kernels
- Optimize messaging interactions via well-known techniques (batching)

Scheduling:

- Implementing a correct scheduler for heterogeneous architectures is a big challenge!
- Even more so for micro-kernels.

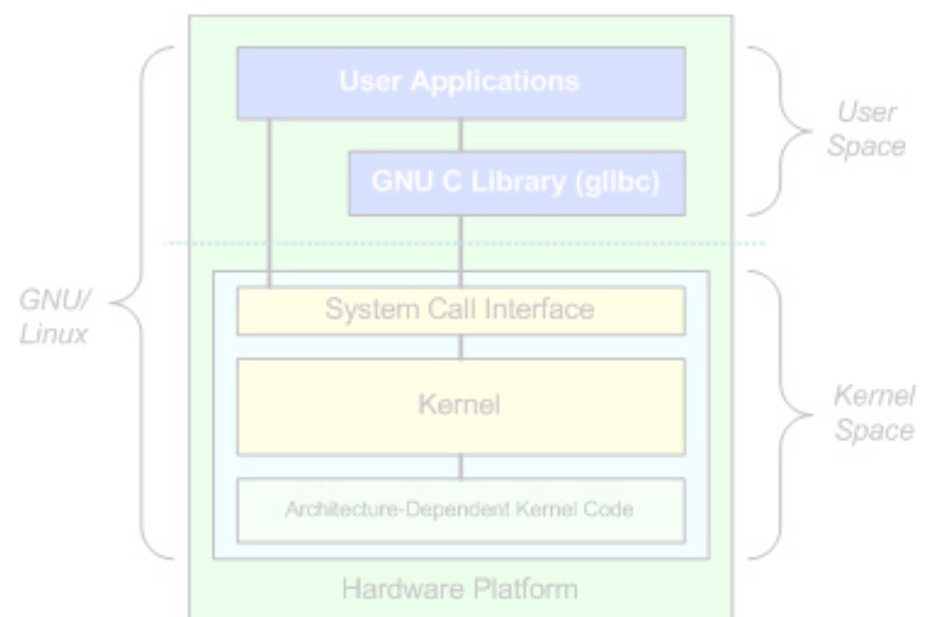
Cache-coherence vs Message-passing:

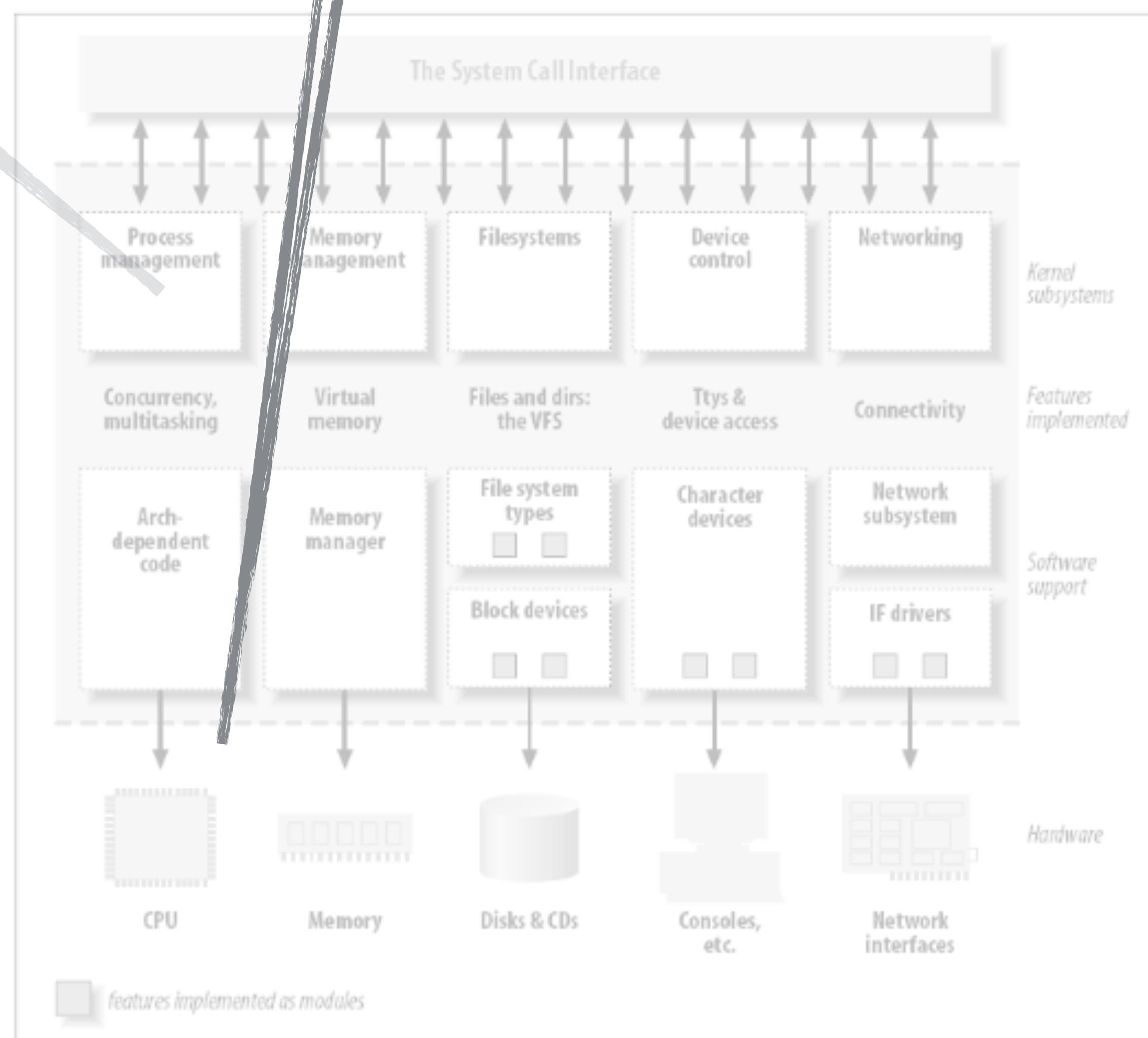
- Abstract and optimize for the fastest approach provided by HW.
- The programming model of the kernel matters!
- Treat them not only as alternatives but things that can be used in parallel!

Linux Kernel

Completely Fair Scheduler

Process Isolation

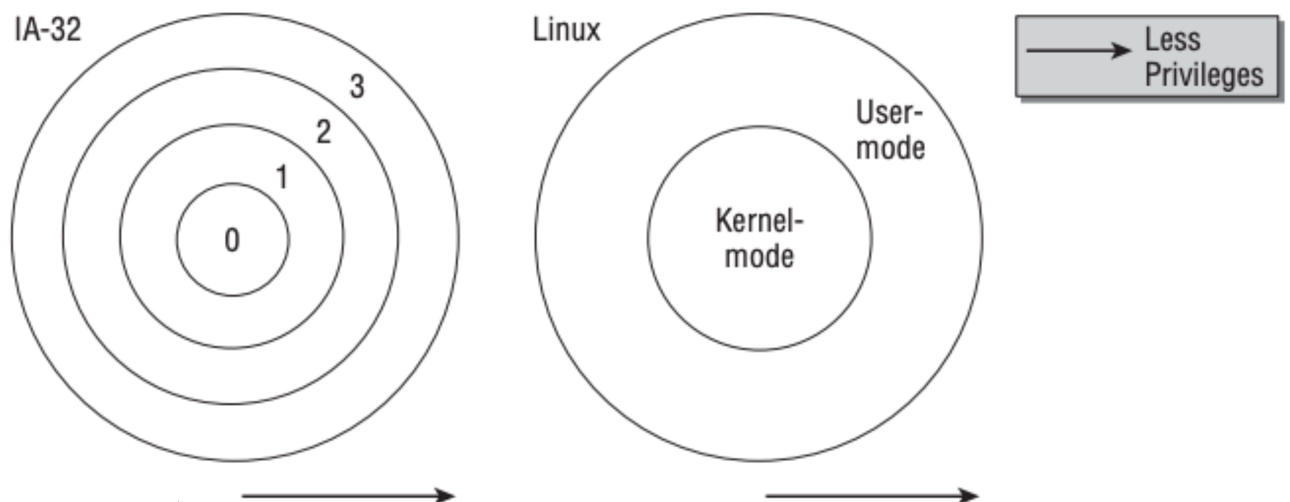




An OS kernel is inherently stateful!

https://www.engineersgarage.com/tutorials/introduction-linux-part-715

Linux Kernel
— Process Isolation —

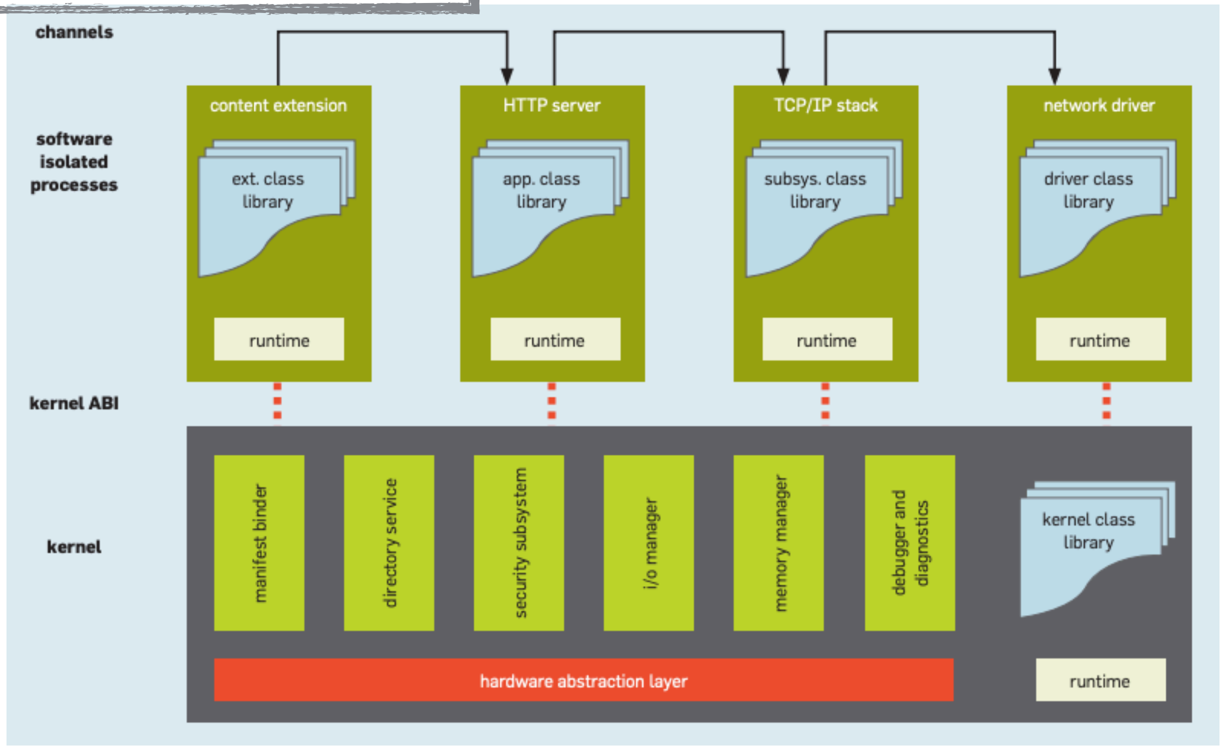


Hardware-based Protection overhead.
MMU ➪ Control Register 0 (x86)

Mauerer, W., 2010. *Professional Linux kernel architecture*. John Wiley & Sons.

Singularity

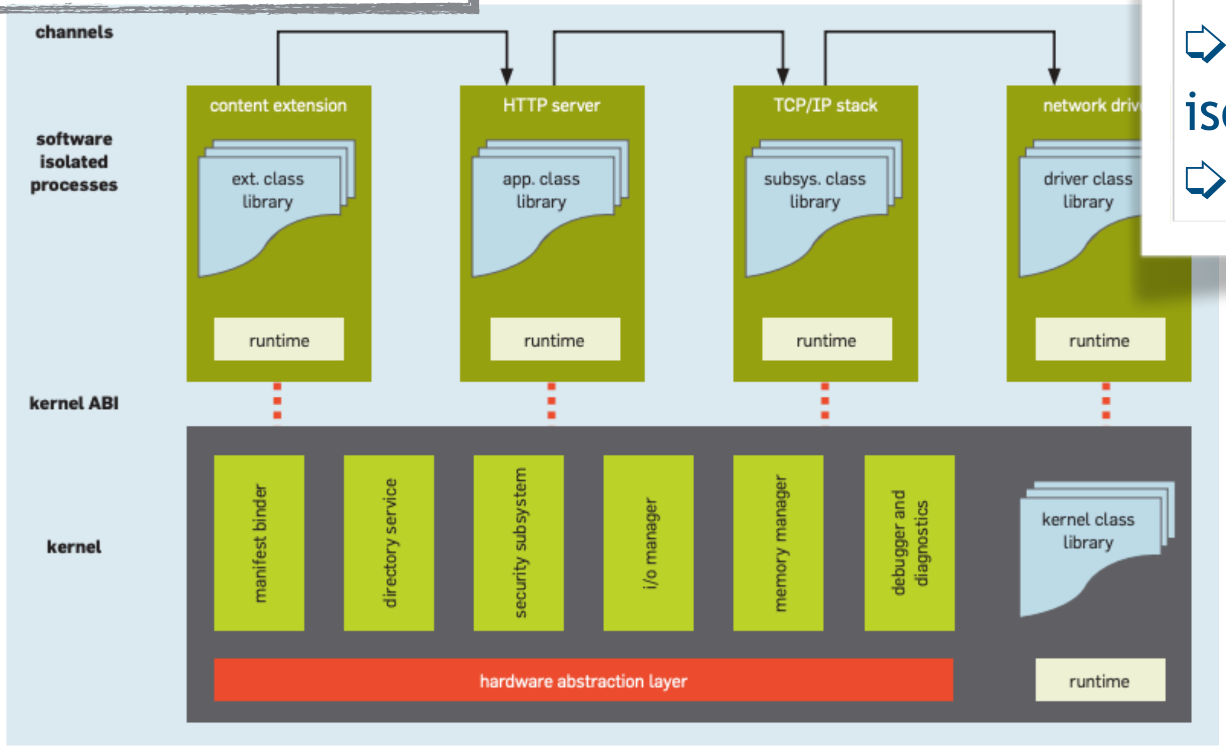
Microkernel design.



Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

Singularity

Microkernel design.



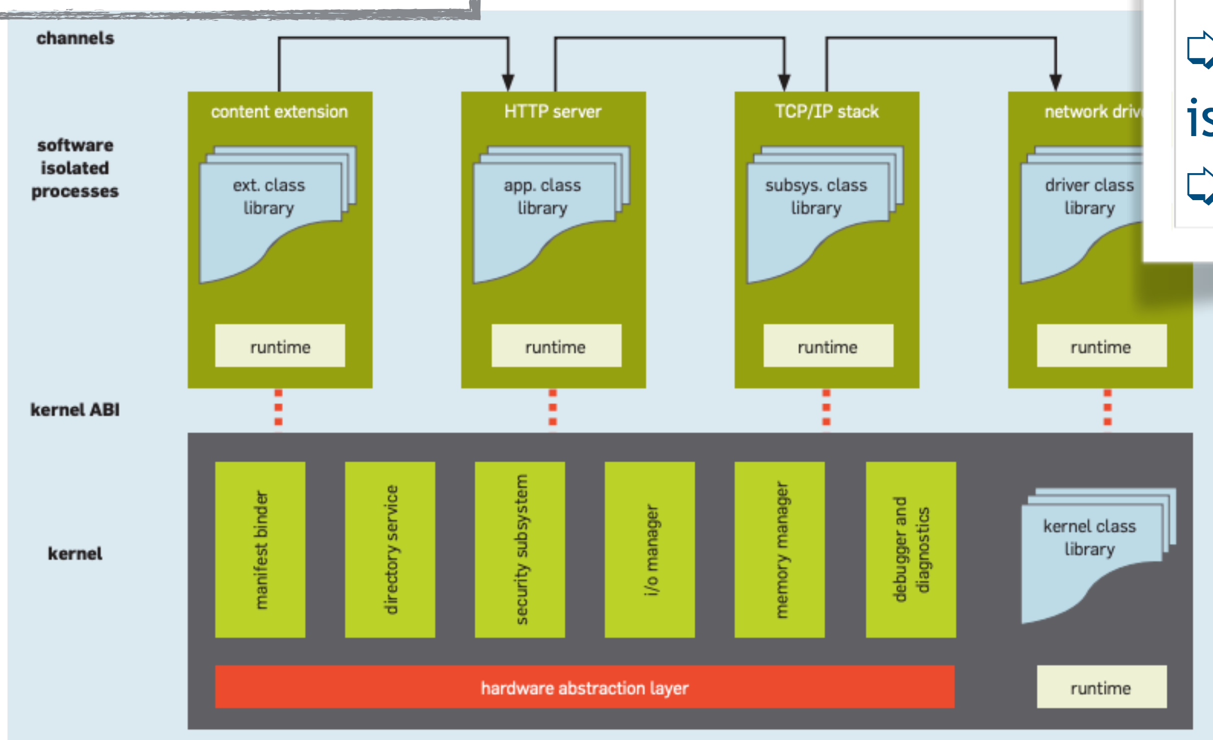
Highlights:

- ownership/linear types to construct software isolated processes
- contracts (session types) to verify across processes

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

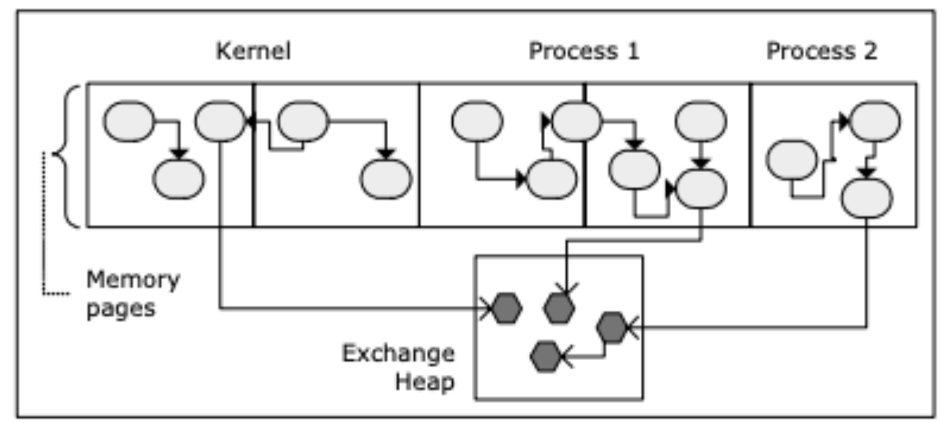
Singularity

Microkernel design.



Highlights:

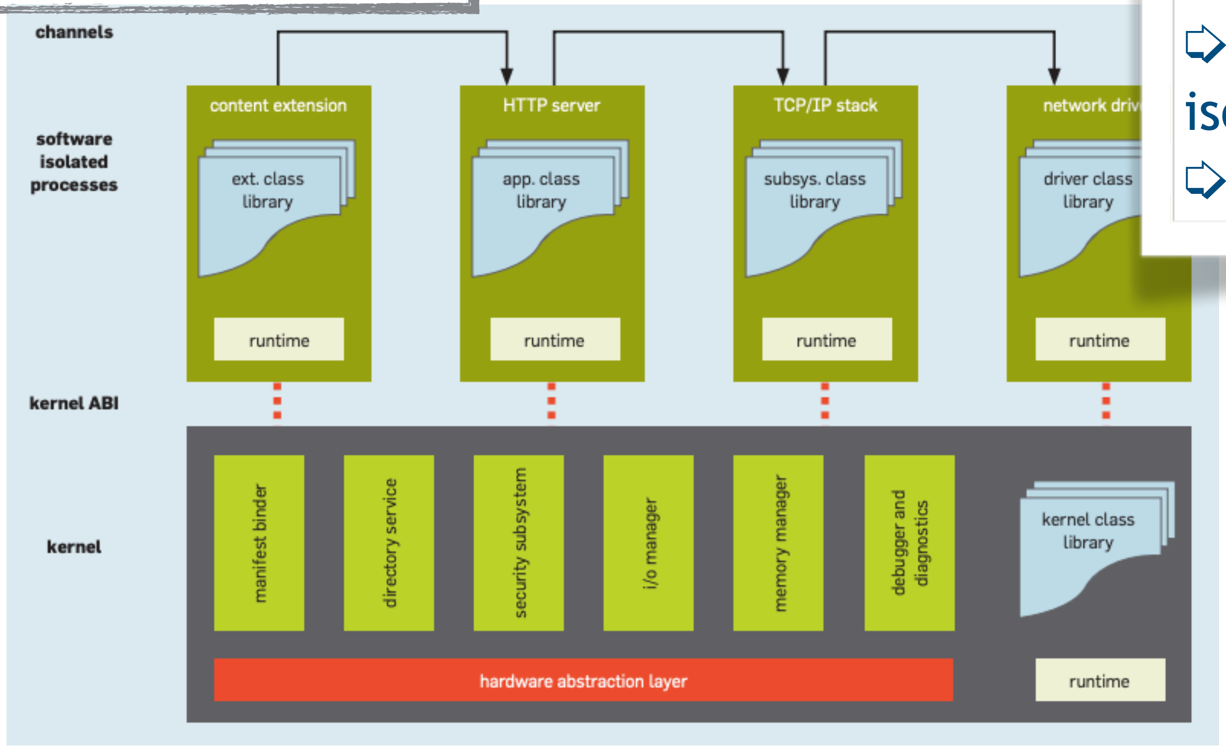
- ➪ ownership/linear types to construct software isolated processes
- ➪ contracts (session types) to verify across processes



Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

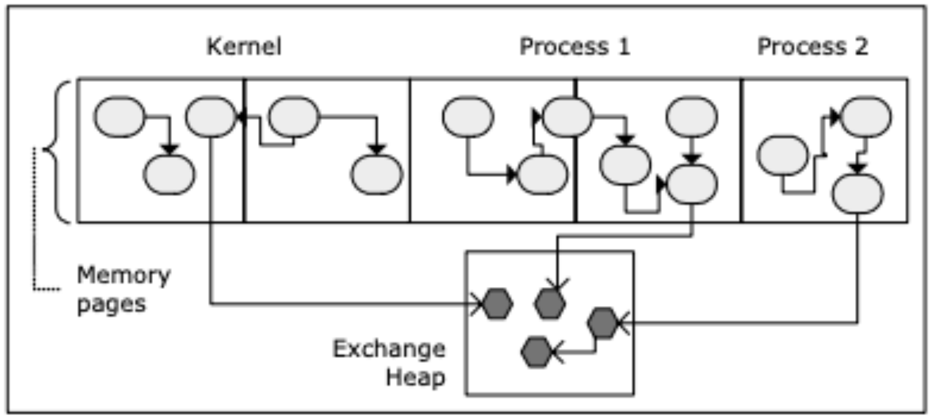
Singularity

Microkernel design.



Highlights:

- ownership/linear types to construct software isolated processes
- contracts (session types) to verify across processes



Analysis:

- no hardware protection required!
- zero-copy messaging!
- SIPs rely on correctness of trusted, unsafe code (runtime & gc)

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

Singularity

Comparison possible!

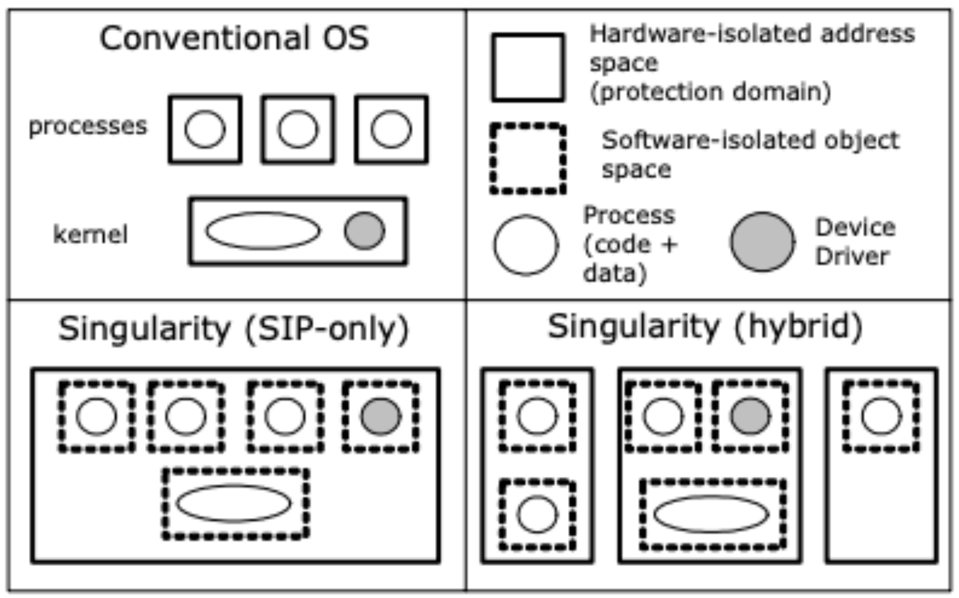
Analysis:

- ➪ no hardware protection required!
- ➪ zero-copy messaging!
- ➪ SIPs rely on correctness of trusted, unsafe code (runtime & gc)

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

Singularity

Comparison possible!



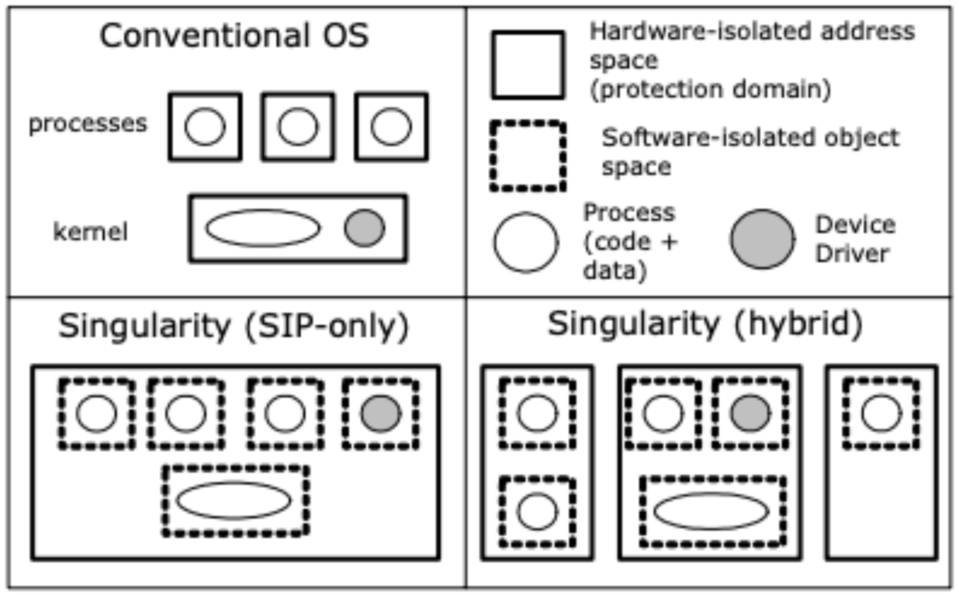
Analysis:

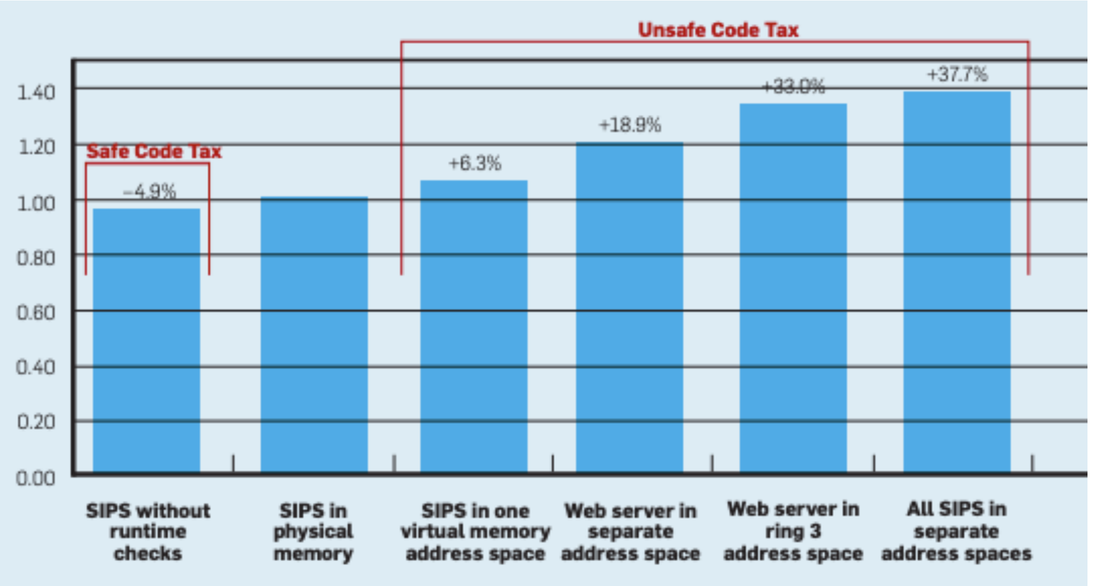
- no hardware protection required!
- zero-copy messaging!
- SIPs rely on correctness of trusted, unsafe code (runtime & gc)

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

Singularity

Comparison possible!





Analysis:

- no hardware protection required!
- zero-copy messaging!
- SIPs rely on correctness of trusted, unsafe code (runtime & gc)

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. and Larus, J., 2006, October. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness* (pp. 1-10). ACM.
Hunt, G.C. and Larus, J.R., 2007. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, *41*(2), pp.37-49.
Larus, J. and Hunt, G., 2010. The singularity system. *Communications of the ACM*, *53*(8), pp.72-79.

Take-away Points

Efficient communication and process isolation:

➪ Ownership types and Linear types are your friends!

Cross-application verification:

➪ Check out Session Types!

Open challenges:

➪ Trusted code verification.

➪ Support for applications written in other languages.

Jung, R., Jourdan, J.H., Krebbers, R. and Dreyer, D., 2017. RustBelt: Securing the foundations of the Rust programming language. *Proceedings of the ACM on Programming Languages*, *2*(POPL), p.66.

Bernardy, J.P., Boespflug, M., Newton, R.R., Peyton Jones, S. and Spiwack, A., 2017. Linear Haskell: Practical linearity in a higher-order polymorphic language. *Proceedings of the ACM on Programming Languages*, *2*(POPL), p.5.

Jespersen, T.B.L., Munksgaard, P. and Larsen, K.F., 2015, August. Session types for Rust. In *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming* (pp. 13-22). ACM.