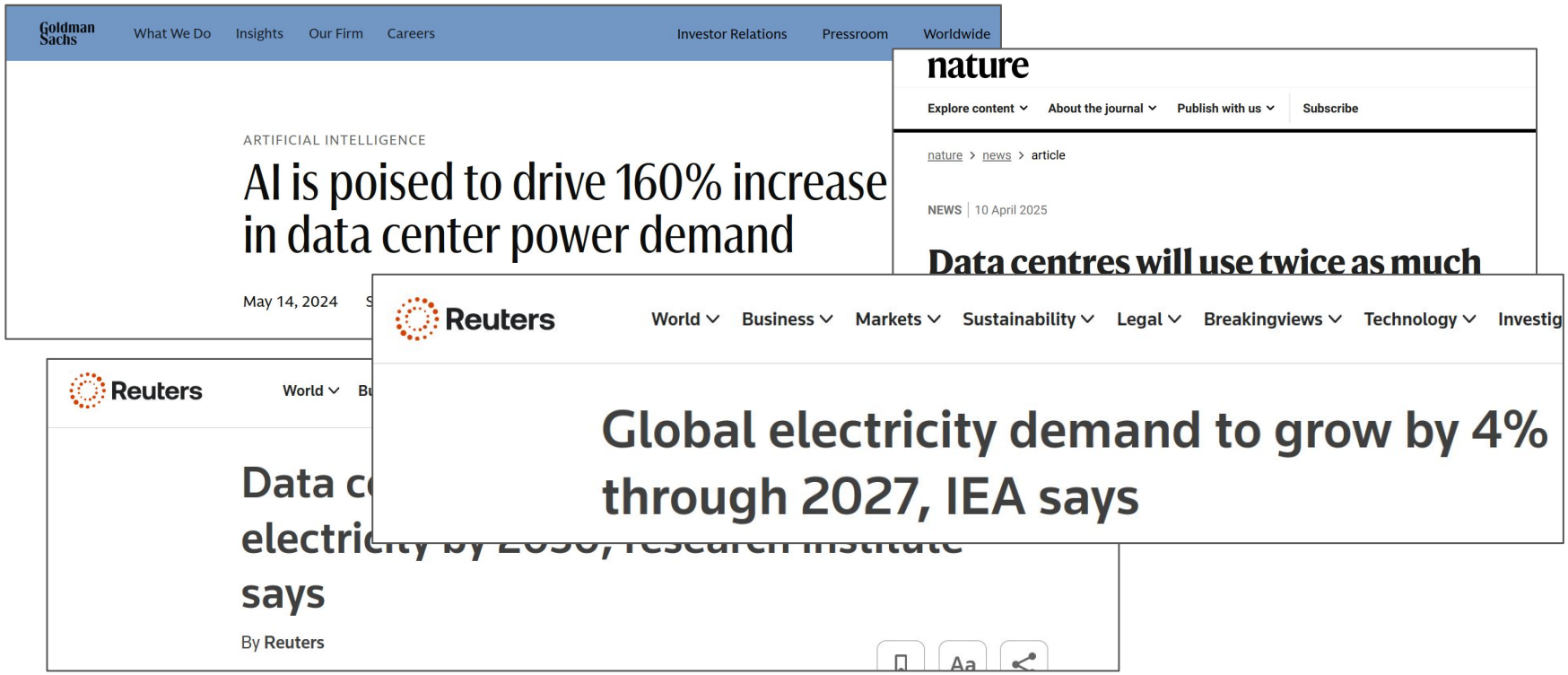
# The Case for Energy Clarity

Fan Chung, Henry Kuo, George Candea

EPFL

# Energy Usage is Growing Dramatically



# Energy Consumption is not Transparent

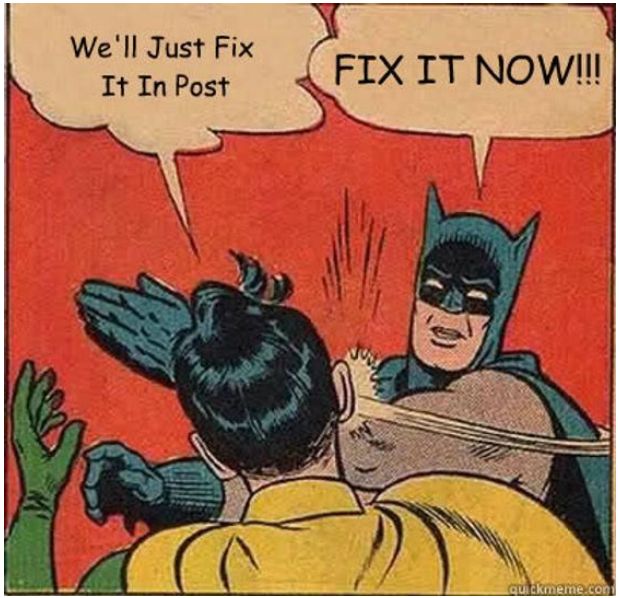

**We need energy clarity!**

# Energy Clarity

- Ability to **reason** about energy for all possible inputs:
  - **What** consumes energy and **how much**
  - How the consumption **changes** with the inputs
  - ...

# Energy Clarity Must Come Before Code

- Energy clarity shouldn’t come as an afterthought.
- Goal: Express, plan, and program energy use **before** the implementation
  - “A call to `LZ4_compress_default()` function should take no more than 35 microjoules, because it's called a lot”
  - “If on the given hardware `LZ4_compress_default()` consumes more than that energy, then call `LZ4_compress_fast()` instead”




**But how can we achieve energy clarity?**

# Energy Interfaces!

# Having Energy Interfaces Like Having Functional Interfaces

- **Functional interfaces** are introduced in the *1960s* to concisely describe a module's functionality

- Imagine if we left functional clarity to be an afterthought…

# Having Energy Interfaces Like Having Functional Interfaces

```
struct list_head *head, **tail = &head;
for (;;) {
    if (cmp(priv, a, b) <= 0) {
        *tail = a;
        tail = &a->next;
        a = a->next;
        if (!a)
            *tail = b, break;
    } else {
        *tail = b;
        tail = &b->next;
        b = b->next;
        if (!b)
            *tail = a, break;
    }
}
return head;
```

```
/*
 * Returns a list organized in an intermediate
 * format suited to chaining of merge() calls:
 * null-terminated, no reserved or sentinel
 * head node, "prev" links not maintained.
 */
static struct list_head *merge(void *priv,
    list_cmp_func_t  cmp, struct list_head *a,
    struct list_head *b)
```

- Functional interfaces are key to functional clarity in building systems

**Energy interfaces are key to energy clarity!**

# Energy Interfaces of an ML Web Service

*takes the input of the original function*

```
def E_ml_service_handle(request):
    max_response_len = 1024
    if request_hit: # ECV: request found in cache
        return E_cache_lookup(request.image, max_response_len)
    else:
        return E_cnn_forward(request.image)
```

ECVs = random vars to describe factors not in the input

# Energy Interfaces of an ML Web Service

```
def E_cache_lookup (key, response_len):
    # ECV:  local_cache_hit - cache hit in current node
    return 5 if local_cache_hit else 100 * response_len # (Microjoules)
```

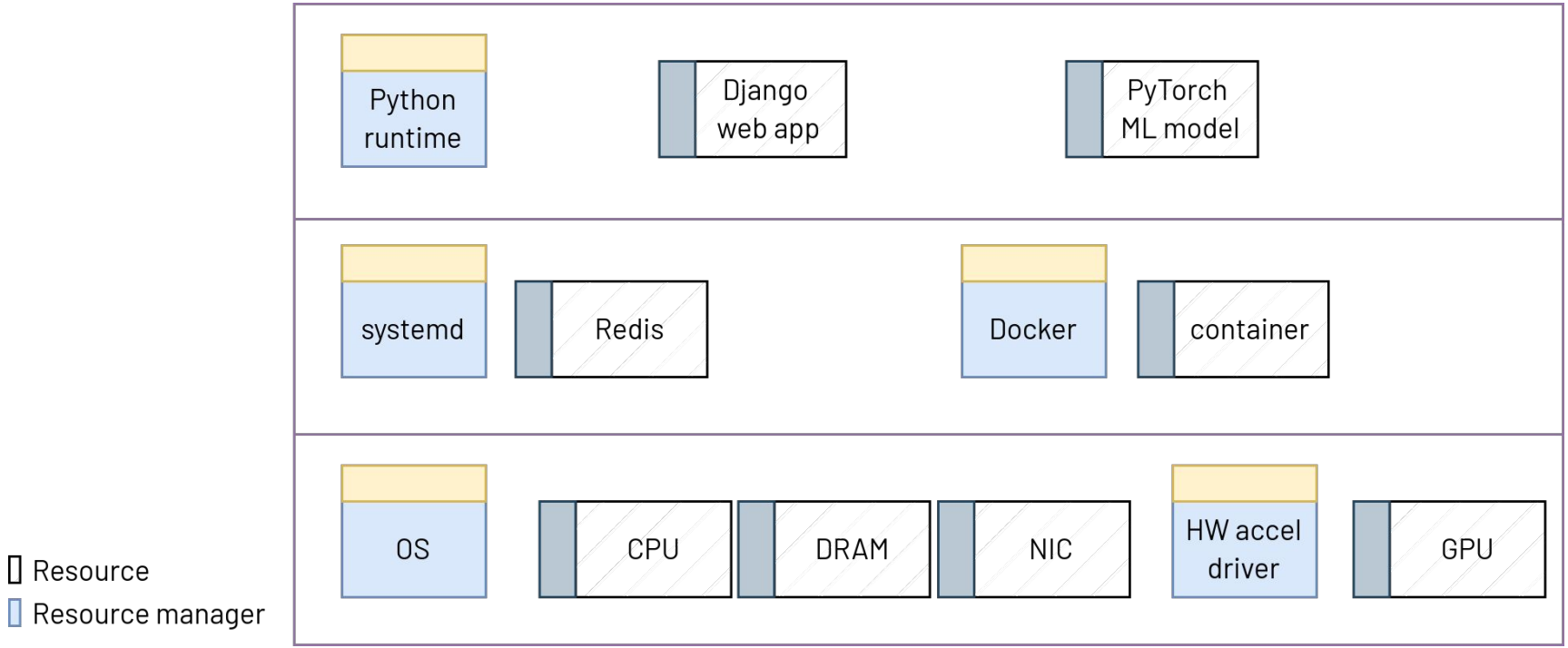
```
def E_cnn_forward (image):
    n_embedding = 256
    n_zeros = image.count(0)
    return (8 * E_conv2d(image.size() - n_zeros) +
            8 * E_relu(n_embedding) +
            16 * E_mlp(n_embedding))
```

# Programs as Energy Interfaces

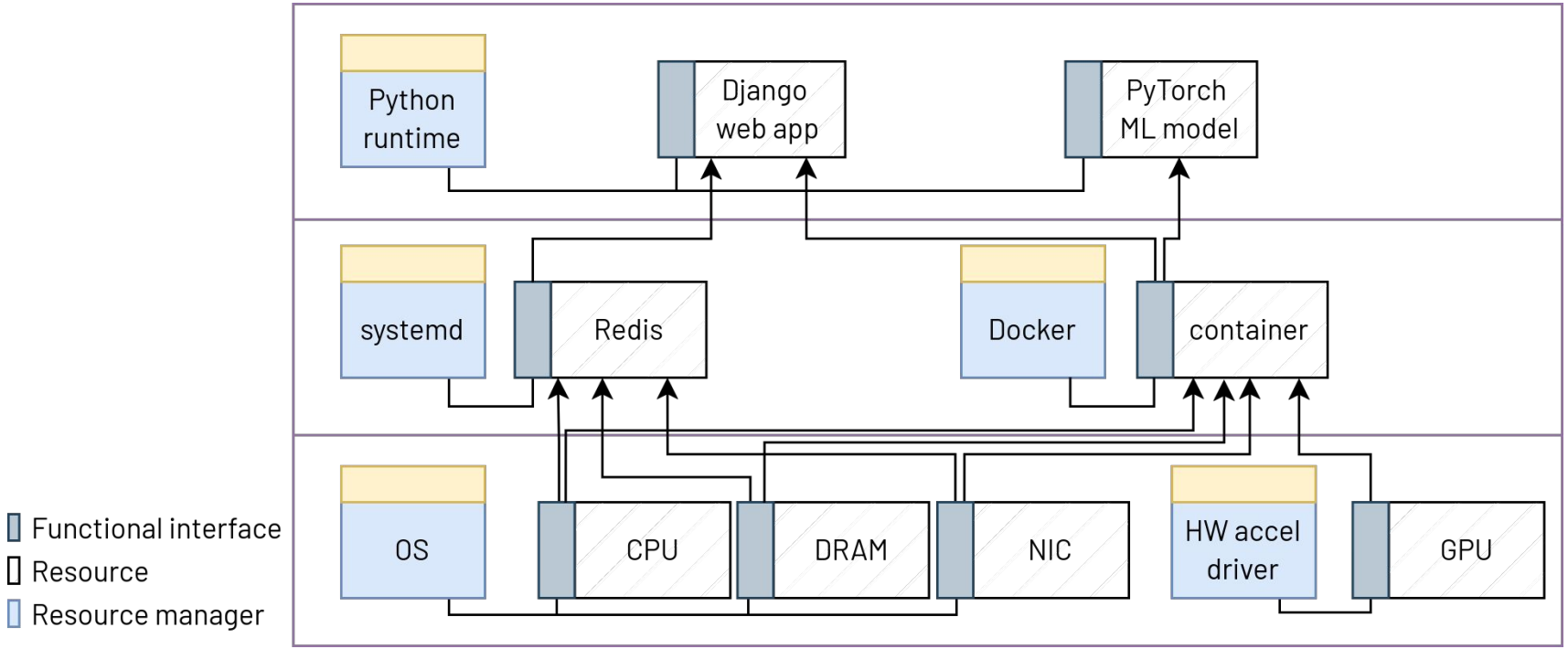
- Energy interfaces are **programs** that describe how much energy a module will use for any possible input.
  - Appeals to programmers' intuition
  - Executable by programs or tools
  - Precisely describes energy behavior
  - Can express any possible energy behavior

**Make energy programmable!**

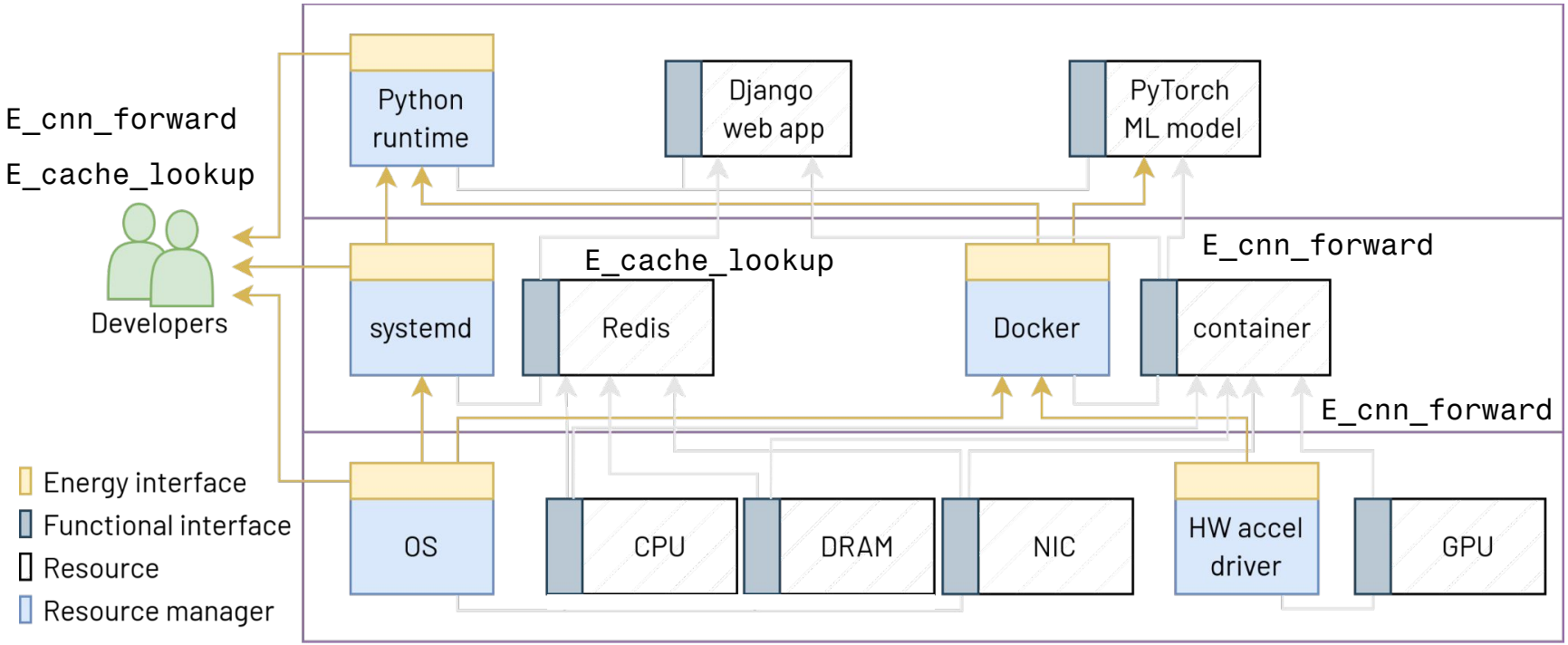
# Energy Interfaces Exported by Resource mgrs



# Energy Interfaces Exported by Resource mgrs



# Energy Interfaces Exported by Resource mgrs



# Energy Interface → Code


```python
def E_cache_lookup (key, response_len):
    # ECV:  local_cache_hit - cache hit in current node
    return 5 if local_cache_hit else 100 * response_len # (Microjoules)
```



```c
const char* cache_lookup (const char* key) {
    int i = hash(key);
    if (cache[i].key && strcmp(cache[i].key, key) == 0)
        return cache[i].value;

    const char* val = cache_remote_fetch(key);
    cache[i] = (CacheEntry){ key, val };
    return val;
}
```

# Code → Energy Interface

Changes: Fetching compressed data and decompressing it locally

```
-     const char* val = cache_remote_fetch(key);
-     cache[i] = (CacheEntry){ key, val };
+     const char* compressed_data = cache_remote_fetch_compressed(key);
+     const char* val = decompress_payload(compressed_data);
+     cache[i] = (CacheEntry){ key, value };
      return val;
  }
```



```
def E_cache_lookup(key, response_len):
     # ECV:  local_cache_hit - cache hit in current node
-    return 5 if local_cache_hit else 100 * response_len       # (Microjoules)
+    return 5 if local_cache_hit else (70 + 10) * response_len # (Microjoules)
```

# Experiment: GPT-2 Energy Interface

```
def E_GPT_2_inference (wl):
    # ECVs: static_power, energy_per_instr, vram_energy_per_access,
l2_energy_per_access, l1_energy_per_access
    static_energy = static_power * exec_time[wl.batch_sz, wl.precision]
    instruction_energy = energy_per_instr * num_instr[wl.batch_sz, wl.precision]
    vram_energy = vram_energy_per_access * sector_rw[wl.batch_sz, wl.precision]
    l2_energy = l2_energy_per_access * l2_rw[wl.batch_sz, wl.precision]
    l1_energy = l1_energy_per_access * l1_rw[wl.batch_sz, wl.precision]

    return static_energy + vram_energy + l2_energy + l1_energy +
instruction_energy
```

# Preliminary Result

| GPU | Average error | Max error |
|---|---|---|
| RTX4090 | 0.70% | 0.93% |
| RTX3070 | 6.06% | 8.11% |

# Recap

- **Energy clarity**, as the ability to reason about energy
- **Energy interfaces** are the key to achieve energy clarity
  - They are **programs** that describe how much energy a module will use for any possible input.
- **Resource managers** combine and expose the energy interfaces of resources

# Open Questions

- **Modularity**
  - How to divide energy behavior into modules just like functional interfaces?
- **Composability**
  - How to compose the higher-level energy interfaces from lower-level interfaces?
- **Automation**
  - Is it possible to automatically obtain the energy interface from a program?

# Call to Action

- Hardware should export more information about the hardware energy state to software
  - “Energy Counters” equivalent to “Performance Counters”
  - Support for energy measurement with higher frequency, and component-specific granularity.

# How Do Energy Interfaces Differ From Other Methods?

| | Energy Profiling | Black-box Energy Modeling | Energy Interface |
|---|---|---|---|
| Can it predict energy consumption? | ❌ | 🟩🟩🟩 | 🟩🟩🟩 |
| For any module, can we know how much energy it consumes? | 🟩 | 🟩 | 🟩🟩🟩 |
| Can we know how does energy change with inputs? | 🟩 | 🟩 | 🟩🟩🟩 |
| Can developers use it to write energy-efficient code? | 🟩 | 🟩🟩 | 🟩🟩🟩 |
| Cost-efficiency? | 🟩 | 🟩🟩 | 🟩🟩🟩 |
| Feasible to adopt in practice? | 🟩🟩 | 🟩🟩 | 🟩🟩🟩 (🌱 Promising) |

# Thank You for Listening!

# Thank You for Listening!