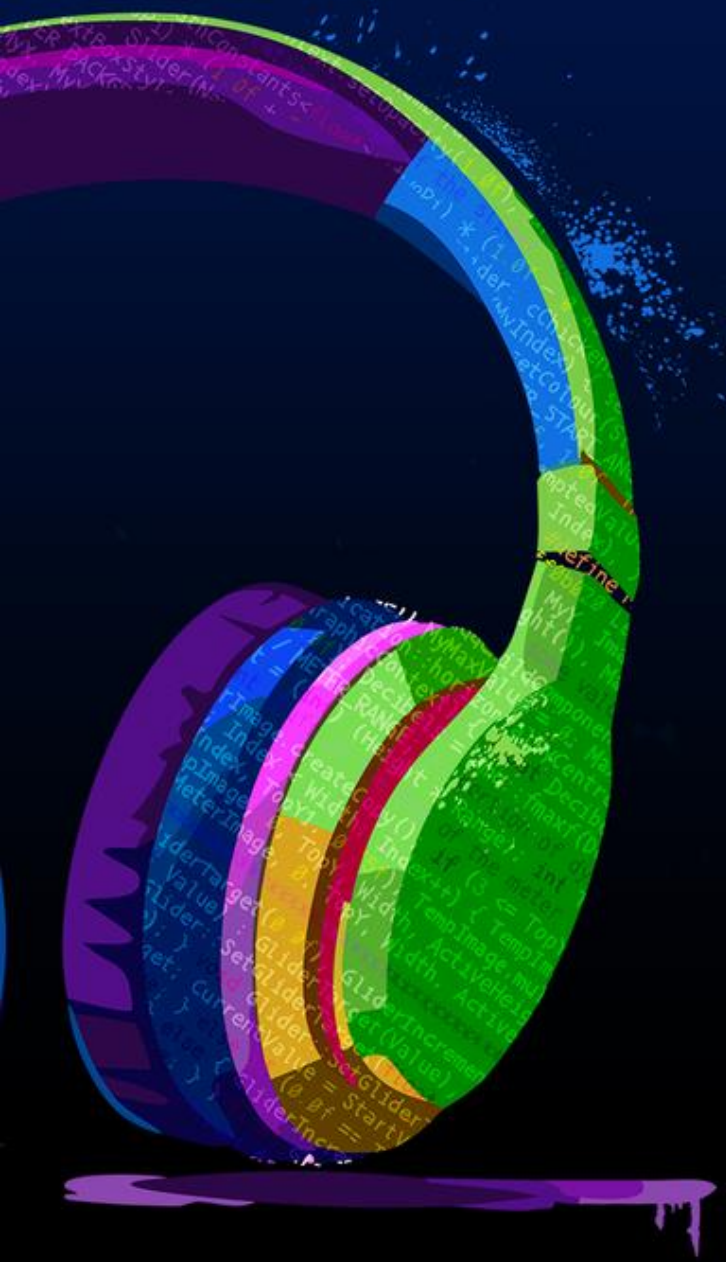


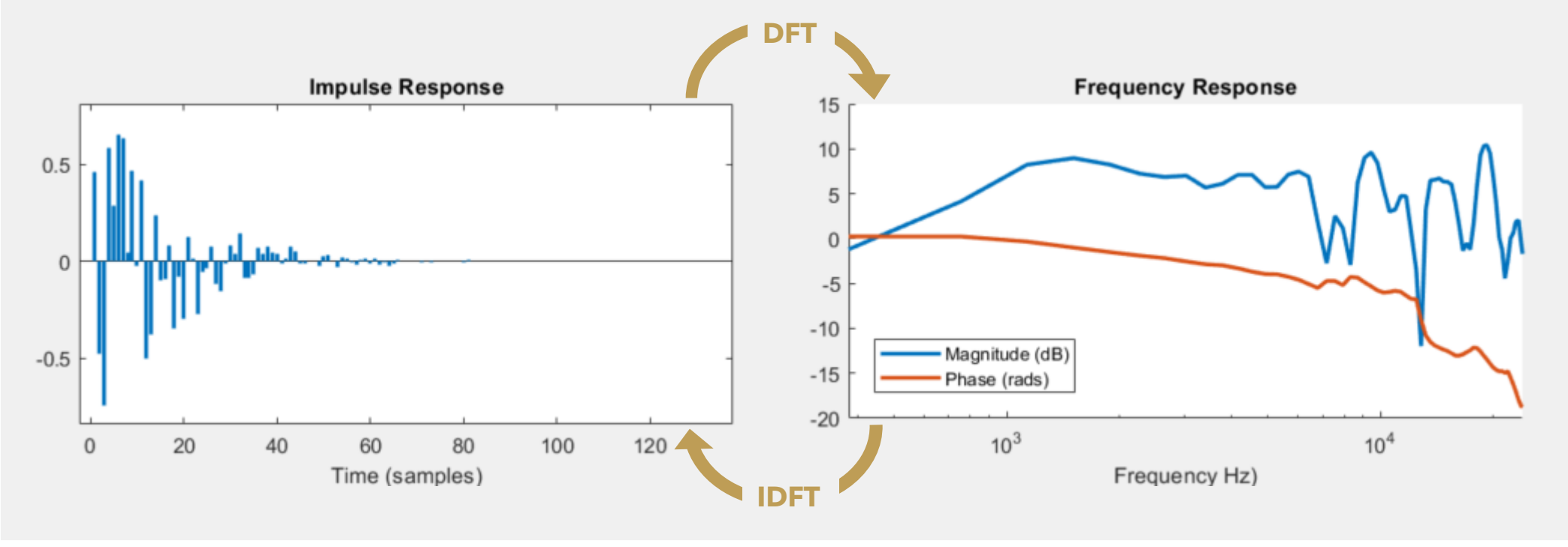
ADC24 Bristol

REAL-TIME FFT CONVOLUTION

HISTORY AND REVIEW

SELIM SHETA

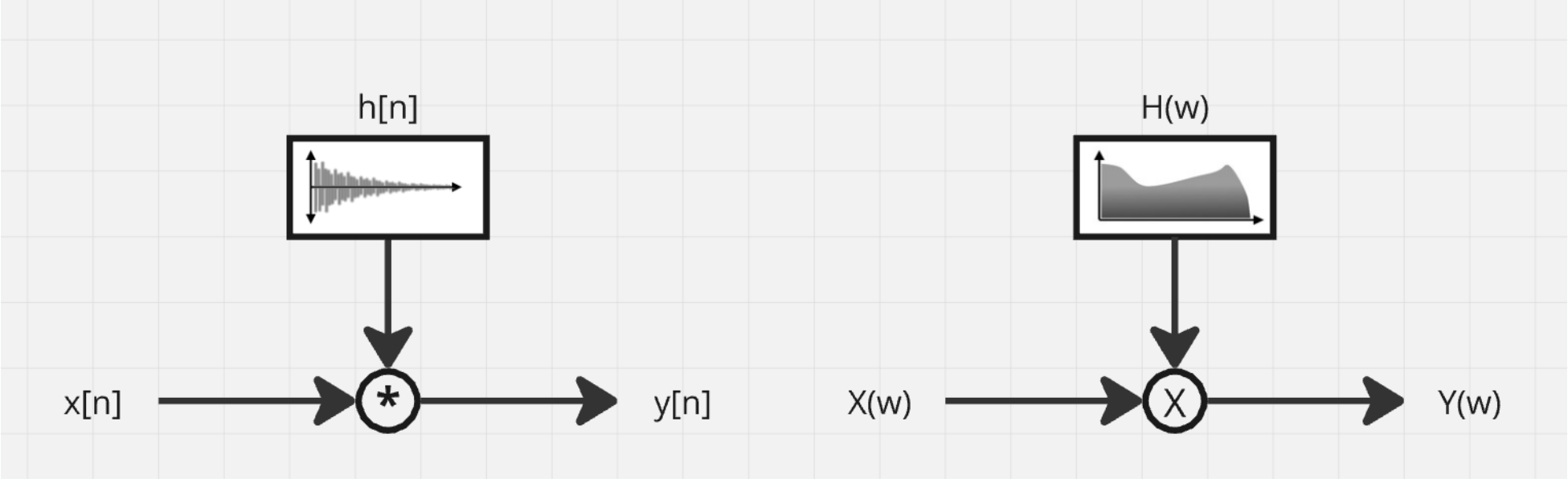
A typical filter* can be described by just its impulse response (IR) and/or frequency response (FR), e.g.:



How do we apply this filter to a signal?

(linear & time-invariant)

- Property of LTI systems: The output is the convolution between the input and the impulse response.
- Convolution in the time-domain is multiplication in the frequency domain (and vice-versa).

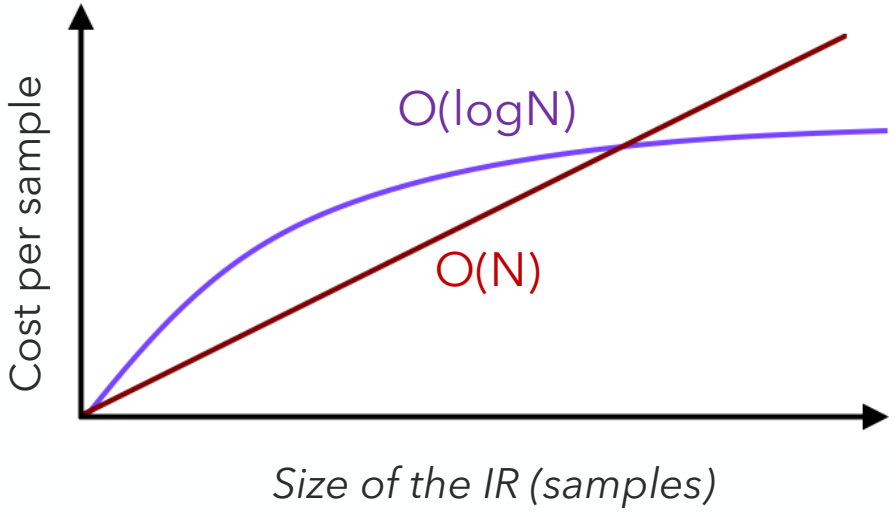


$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k] = \sum_{k=-\infty}^{+\infty} h[k]x[n-k]$$

$$Y(w) = X(w)H(w) = H(w)X(w)$$

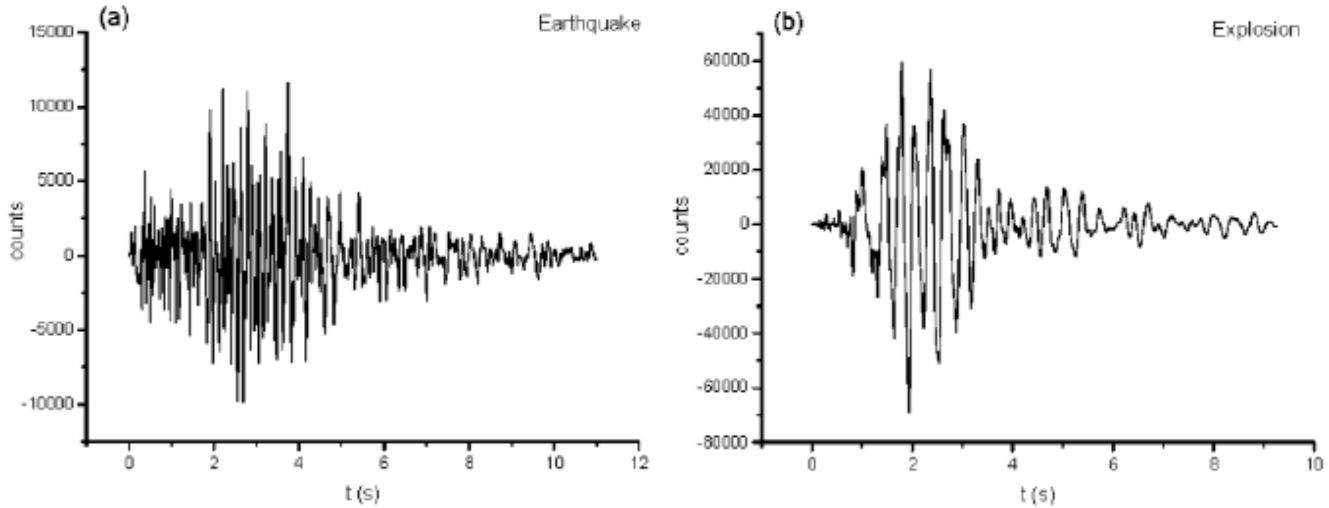
- Time-Domain Convolution (TDC) is O(N). Can we do better in the frequency domain?
  - Computing DFT/IDFT adds overhead
  - Complex operations involve more FLOPs than real operations

```
for (int n = 0; n < blockSize; n++) {
    for (int k = 0; k < irSize; k++) {
        output[n] += input[n-k] * ir[k];
    }
}
```



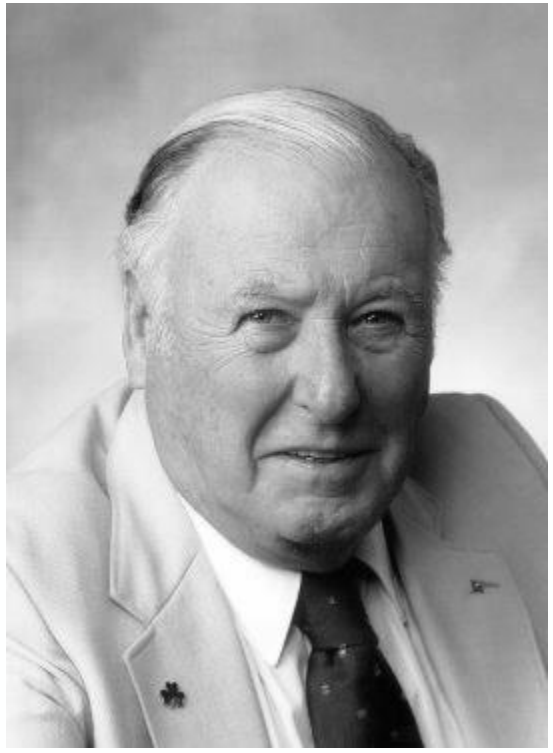
- 1950's: Start of the Nuclear Arms Race
- 1963: Limited Test Ban Treaty, restricting nuclear testing for the USSR, US and UK.
- How do you ensure everyone complies?

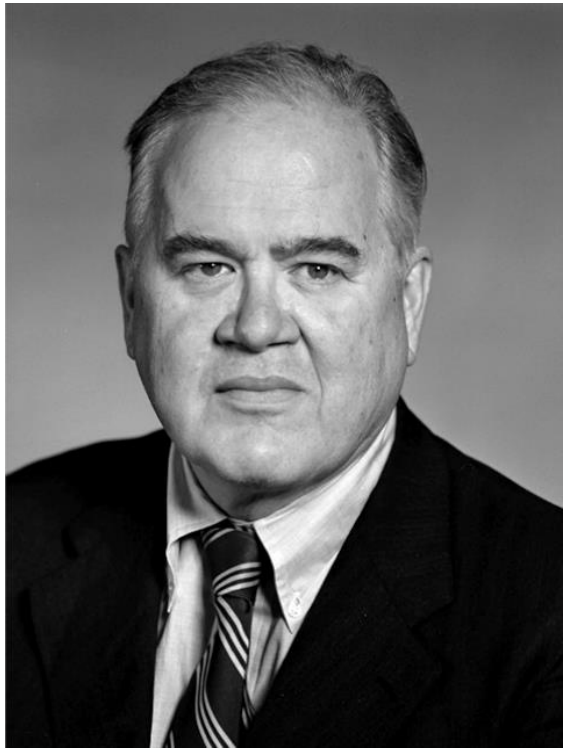
- Atmospheric tests: Measure radioactive isotopes in the air
- Underwater/Underground tests: Detect shockwaves using hydrophones/seismographs…?
  - How to differentiate between a nuclear explosion and a natural geological event?



- Must analyse the frequency content using the discrete Fourier transform.
- Problem: 1960s computers can't compute large DFTs within reasonable time.

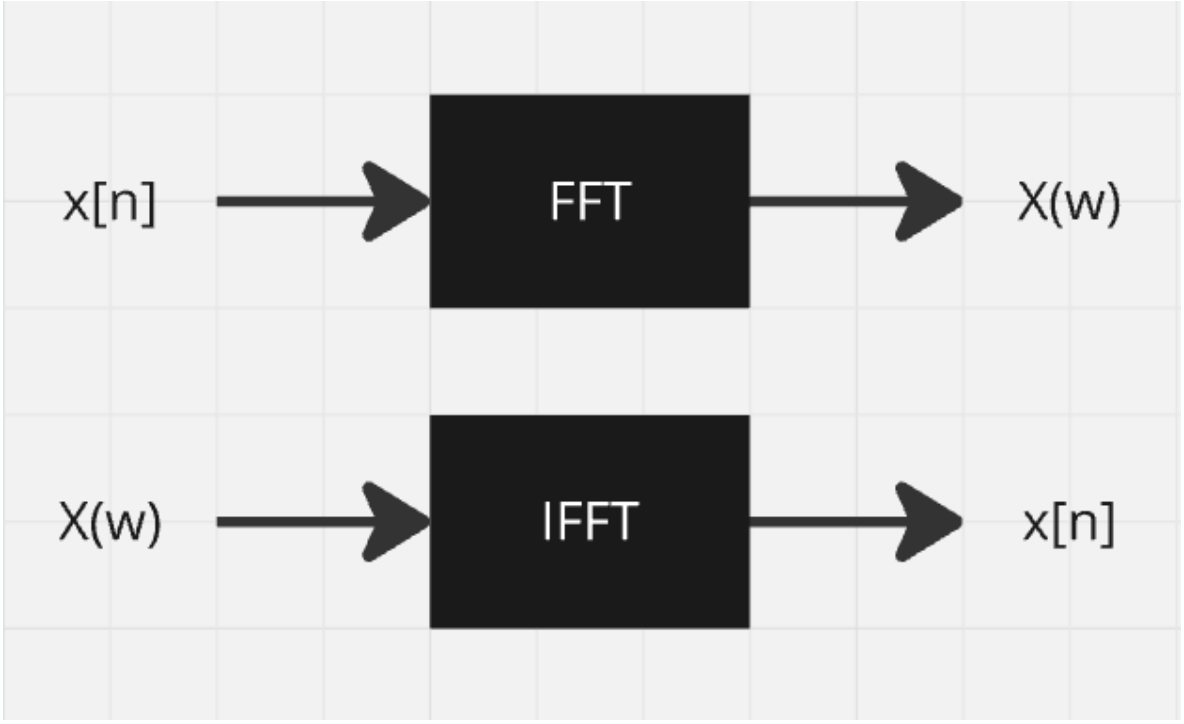
2. Fourier Transform (DFT & FFT)





- Circa 1960 – Tukey begins exploring the math after a meeting with Kennedy
- 1965 - Cooley and Tukey publish their paper describing the modern FFT
- The complexity of the DFT goes from $O(N^2)$ to $O(N\log N)$

2. Fourier Transform (DFT & FFT)



- Returns the frequency-domain representation of a discrete signal
  - Fully reversible
  - Complexity O(NlogN)

3. Overlap-Add & Overlap-Save

- 1966 – Stockham investigates using Cooley & Tuckey's algorithm for convolution
- Mention of splitting one of the two input signals into multiple "sections"
  - i.e. The input can be processed in blocks
- Introduction of the 'Overlap-Add algorithm'

B: Block size
L: IR size
N: FFT size

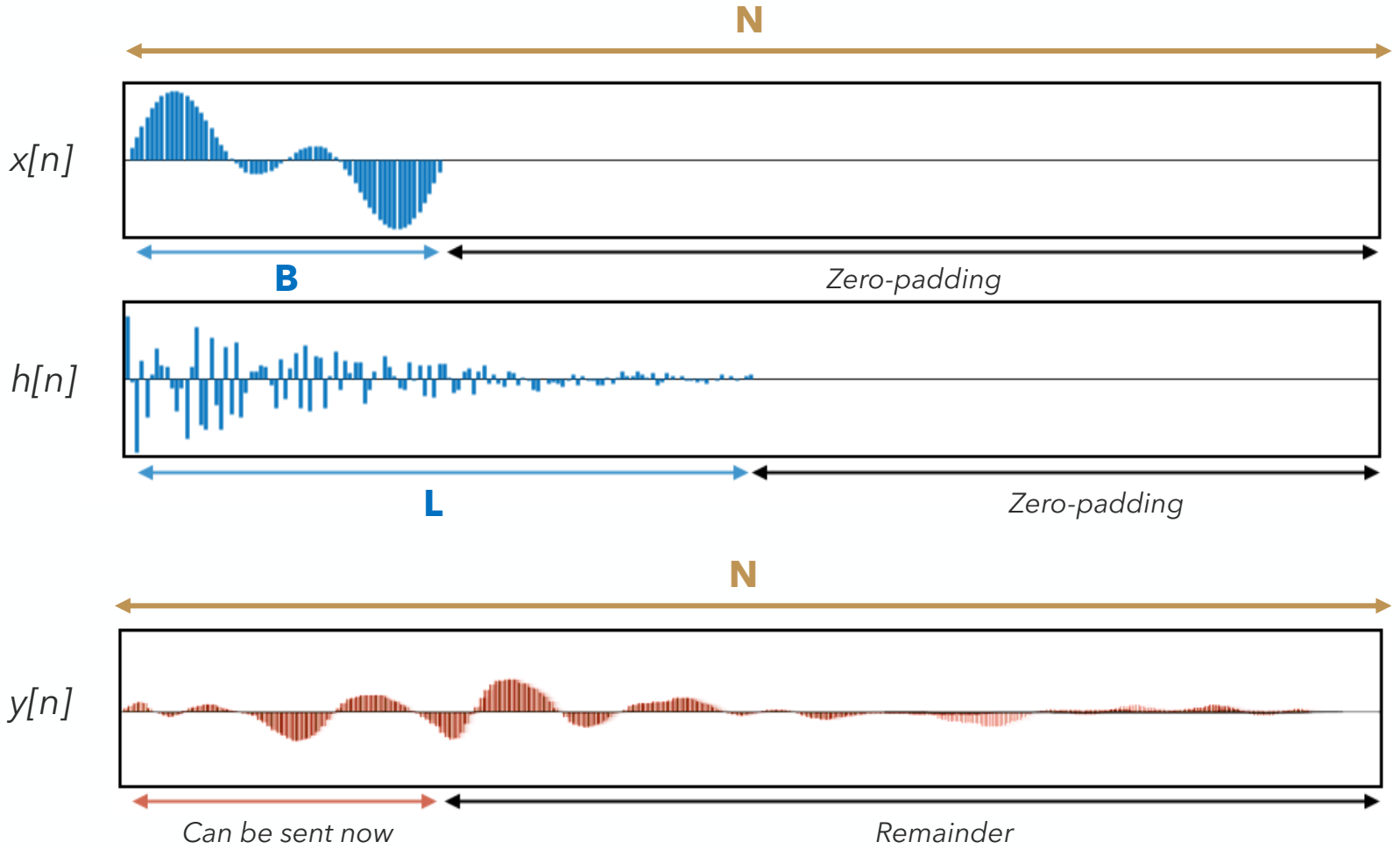
- The output of a convolution between two signals of size N_1 and N_2 is: $\boldsymbol{N_1 + N_2 - 1}$
- So, the size of the spectra must be at least $\boldsymbol{B + L - 1}$ (otherwise aliasing occurs).
- Practical FFT size: $N = 2^{\lceil \log_2 B+L \rceil}$
- We can only send B samples at a time to the output stream... What to do with the rest?

B N

```
paddedInput = zeroPad(input, fftSize);
inputSpectrum = FFT(paddedInput);
for (int i = 0; i < 2 * blockSize; i++) {
    inputSpectrum[i] = inputSpectrum[i] * filterSpectrum[i];
}
fullOutput = IFFT(inputSpectrum);
```

N

B: Block size
L: IR size
N: FFT size



B: Block size
L: IR size
N: FFT size

Preparation

Zero-pad the IR to the target length N
Take the FFT of the padded IR and store it
Initialize an empty buffer of size N

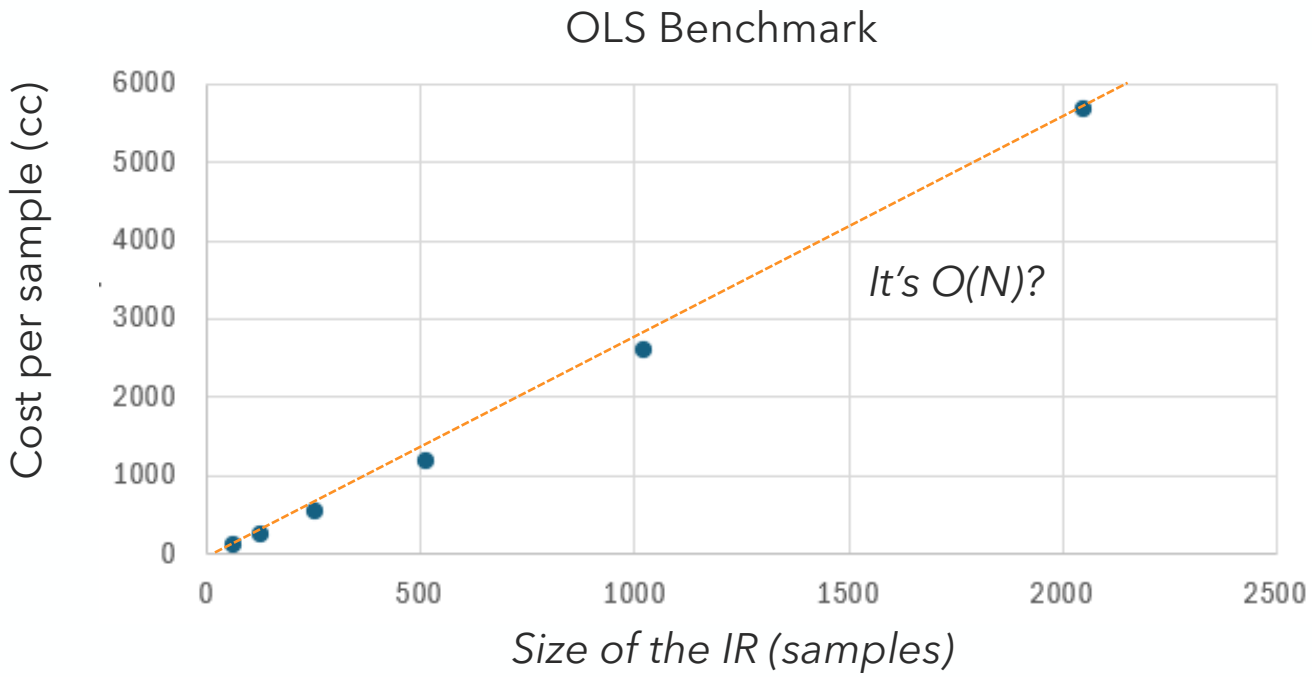
Overlap-Add (OLA)

1. Zero-pad the block to the target length N
2. Perform the frequency-domain convolution*
3. Add the entire convolution result to the buffer
4. Send the first B samples of the buffer to the output stream
5. Clear those samples and shift the buffer

Overlap-Save (OLS)

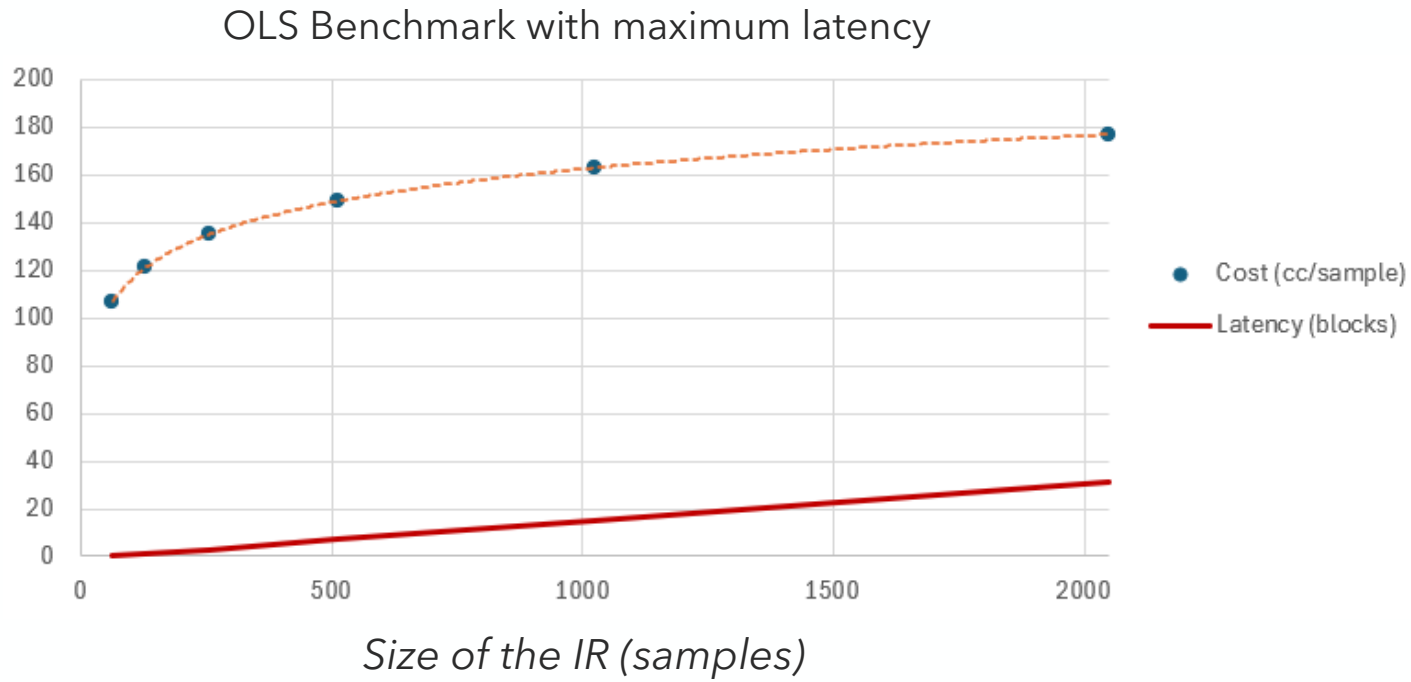
1. Shift the buffer
2. Copy the block to the last B samples of the buffer
3. Perform the frequency-domain convolution*
4. Send the last B samples of the convolution result to the output stream, discard the rest

*
```
inputSpectrum = FFT(input);
for (int i = 0; i < blockSize; i++) {
    outputSpectrum[i] = inputSpectrum[i] * irSpectrum[i];
}
output = IFFT(outputSpectrum);
```

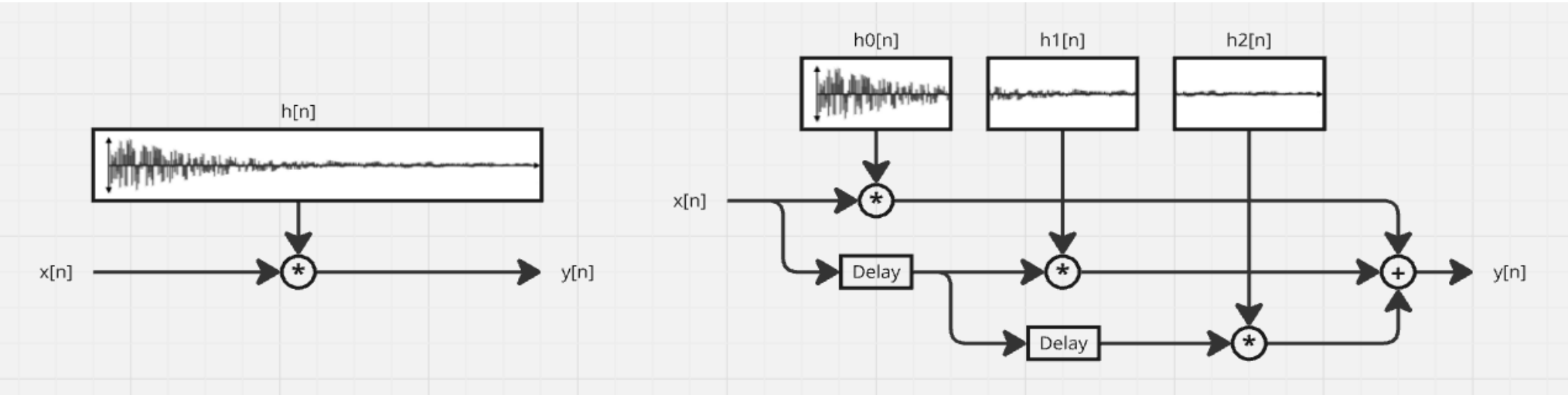

OLS Benchmark
Cost per sample (cc)
6000
5000
4000
3000
2000
1000
0
0
500
1000
1500
2000
2500
Size of the IR (samples)
It's O(N)?

B: Block size
L: IR size
N: FFT size

- OLA and OLS are O(logN) if L ≤ B
- Solution: Latency (process multiple blocks together, increasing the effective block size)
- Ideally, accumulate $\frac{L}{B}$ blocks before performing the convolution
- Without latency, only good for very short IRs ($\leq B$)

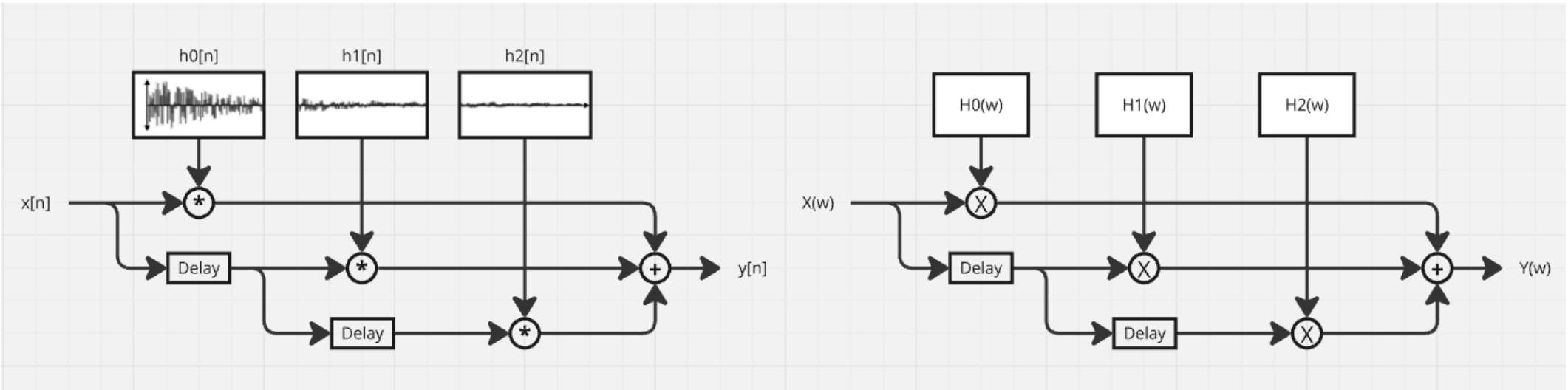


- 1970s: Research is focused on beating the Cooley-Tuckey (radix-2) FFT
- OLA/OLS + latency remains the practical solution for frequency-domain convolution
- ‘Partitioning’ initially refers to splitting the input into blocks (enabling real-time processing)
- 1980’s: We begin partitioning the impulse response as well:



- It also works in the Frequency-Domain.
- A uniform partition scheme is described by two parameters:
  - Part Size P: The length of the chunks in samples.
  - Multiplicity M: The number of parts.
- For an IR of size L, we want $\boldsymbol{P \cdot M > L}$

Is it better to do few big convolutions, or many small convolutions?

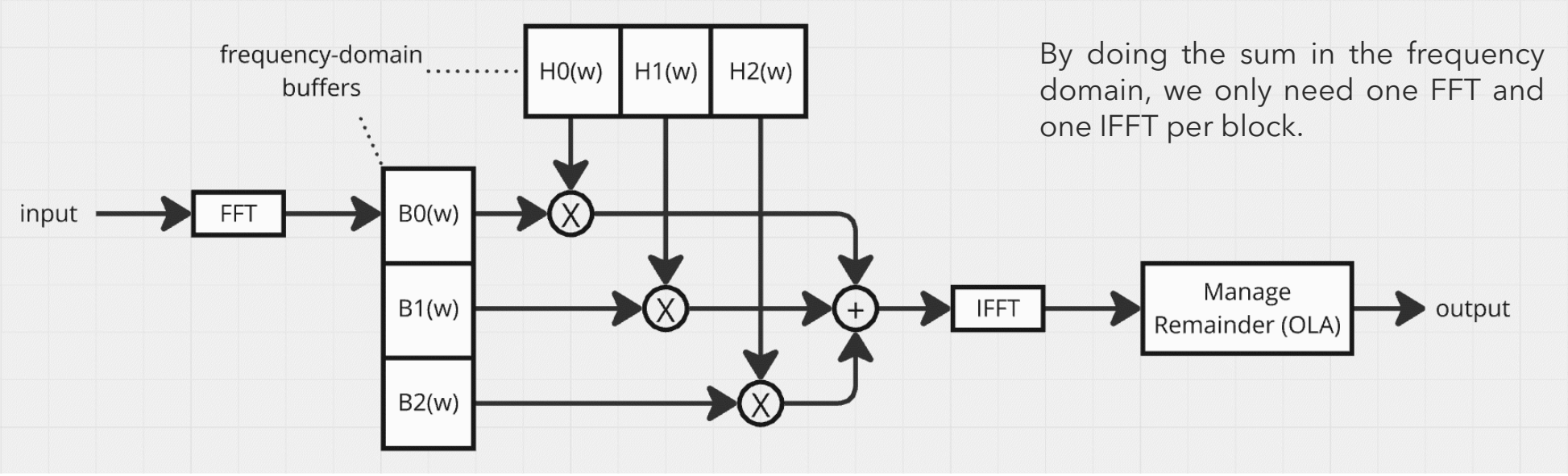


Preparation

Decide a partition scheme [P, M]
Split the IR into chunks of size P
Zero-pad each chunk to size N = 2P
Compute and store the FFT of each padded chunk

Processing

1. Zero-pad the block to the target length N = 2P
2. Compute its FFT & store in a frequency-domain delay line
3. Multiply each IR chunk H_n with the spectrum of block b_n
4. Sum all the convolutions to get the frequency-domain result
5. Take the IFFT of the sum to get the time-domain result
6. Send the first B samples and manage the remainder using OLA



'Uniform Convolver'

B: Block size
L: IR size
P: Part size
N: FFT size (2P)
M: Multiplicity (L/P)

- $C_{FFT}(\mathrm{N}) = k_0 N \log_2(N)$ → cost of computing the FFT of N samples.
- $C_{CMA}(\mathrm{N}) = k_1 N$ → cost of N complex multiply-adds.
- For an IR of size L and a partition scheme [P,M] the total cost* of the convolution is approx.:

$$C = 2\times C_{FFT}(\mathrm{N}) + M\times C_{CMA}(\mathrm{N})$$

$$C = 2k_0 N \log_2(N) + Mk_1 N$$

$$C = 2k_0 2P \log_2(2P) + \frac{L}{P} k_1 2P$$

$$C = 4k_0 P \log_2(2P) + 2k_1 L$$

Hence, we should use the smallest part size possible (P=B in practice).
This means many small convolutions are better than few large ones.

*Ignoring copy/shift operations and assuming we use OLS

B: Block size
L: IR size
T: Latency
P: Part Size
N: FFT size (2P)
M: Multiplicity (L/P)

- Let's introduce latency: P = $(1 + T)B$, where T is the latency in blocks*
- Collect $(1 + T) = \frac{P}{B}$ blocks at time and convolve them all at once.
- From before, the total cost of the convolution is approx.:

$$C = 4k_0 P \log_2(2P) + 2k_1 L$$

But we only need to compute it once every $\frac{P}{B}$ blocks, so the average cost per block is:

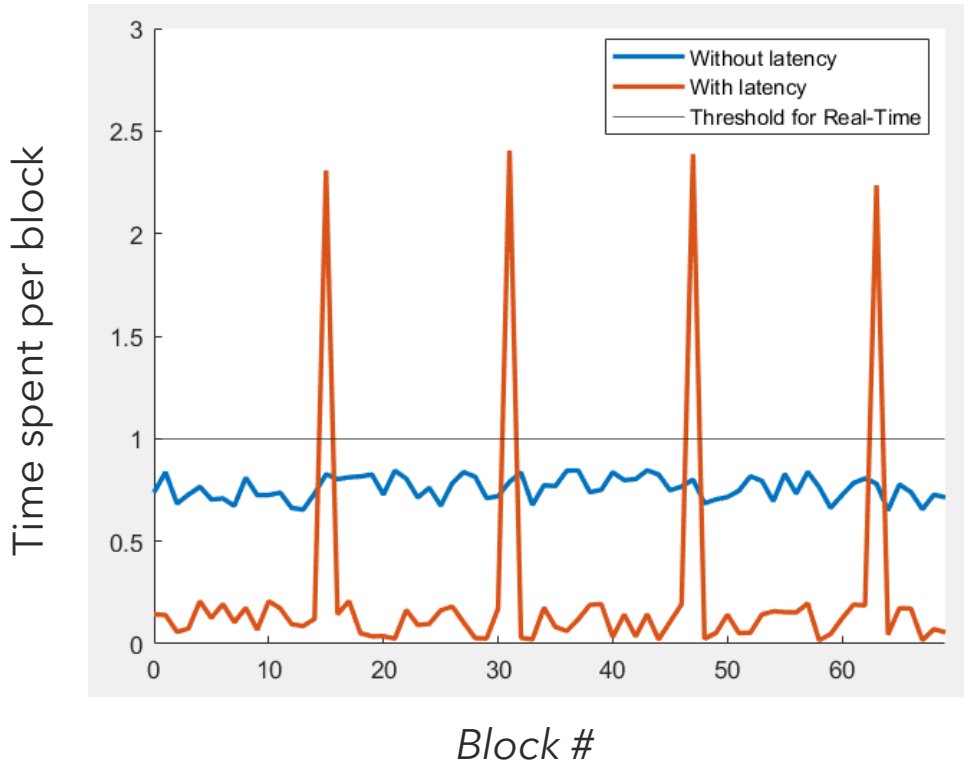
$$\bar{C} = \frac{C}{P/B} = 4k_0 B \log_2(2P) + 2k_1 B \frac{L}{P}$$

And the average cost per sample is:

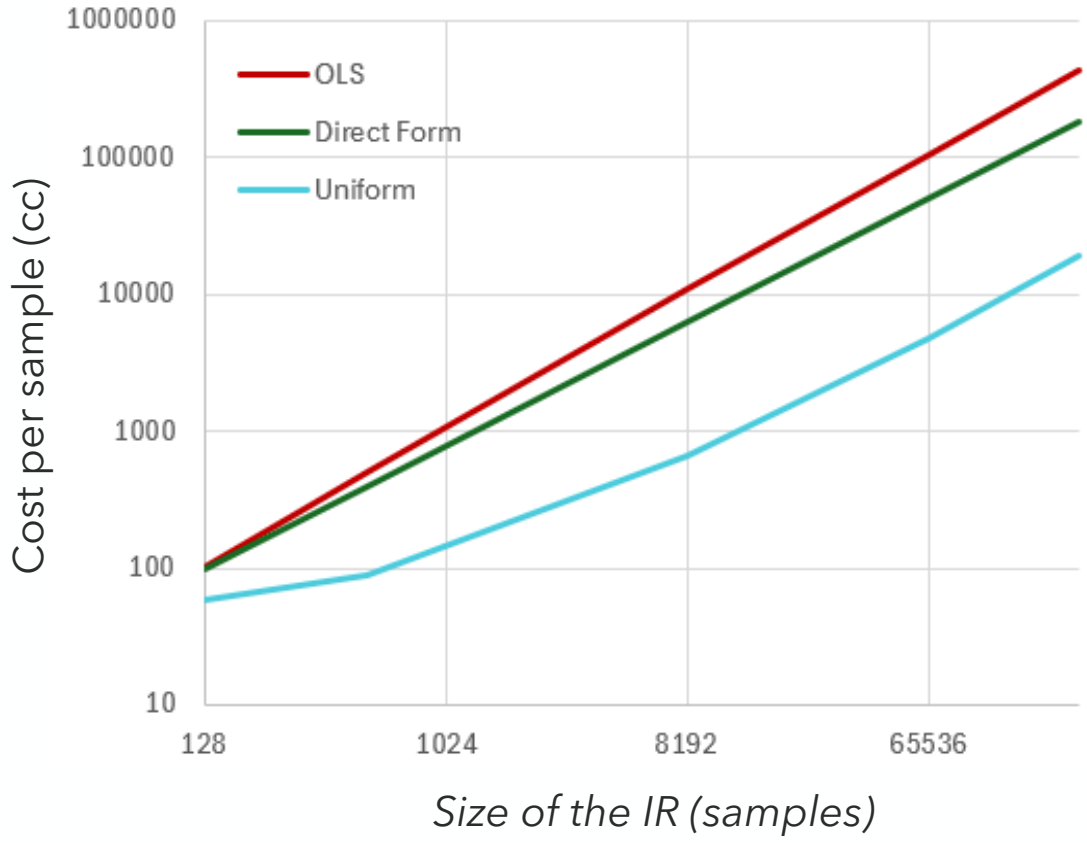
$$\bar{\bar{C}} = \frac{\bar{C}}{B} = 4k_0 \log_2(2P) + 2k_1 \frac{L}{P}$$

*(1+T) should be a power of 2, such that P & N are also powers of 2.

- Any amount of latency helps if $L > 4\frac{k_0}{k1}B$
- While it reduces the average CPU load, the peak load is still important.
- With long IRs, peaks can result in frame drops.



Without latency, only viable for medium IRs ($< 128B$)



1990s: Convolution reverb is in demand -> Need to handle much longer IRs in real-time

Sony DRE-S777 : First convolution reverb hardware
Release Date/Price: 1999, £5870
Max IR length: 5.5 seconds

5. Non-Uniform Partitions

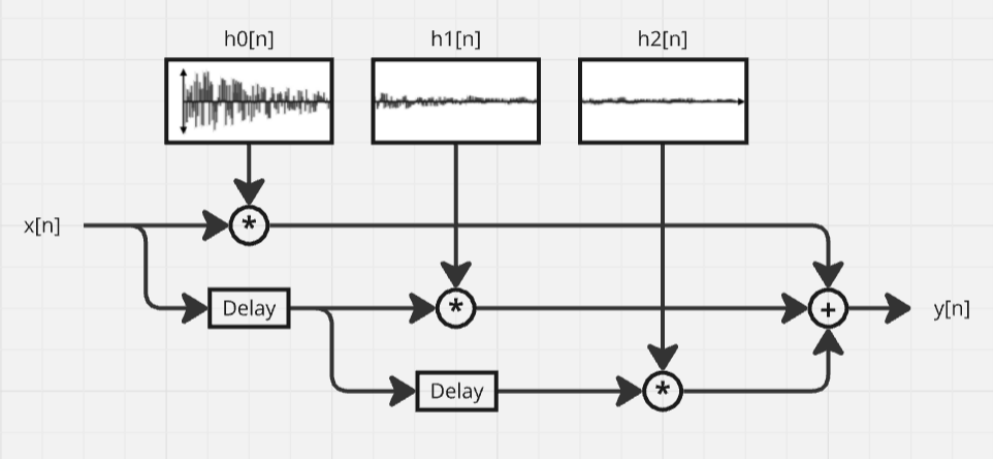
Egelmeers & Sommen (1994), Gardner (1995)

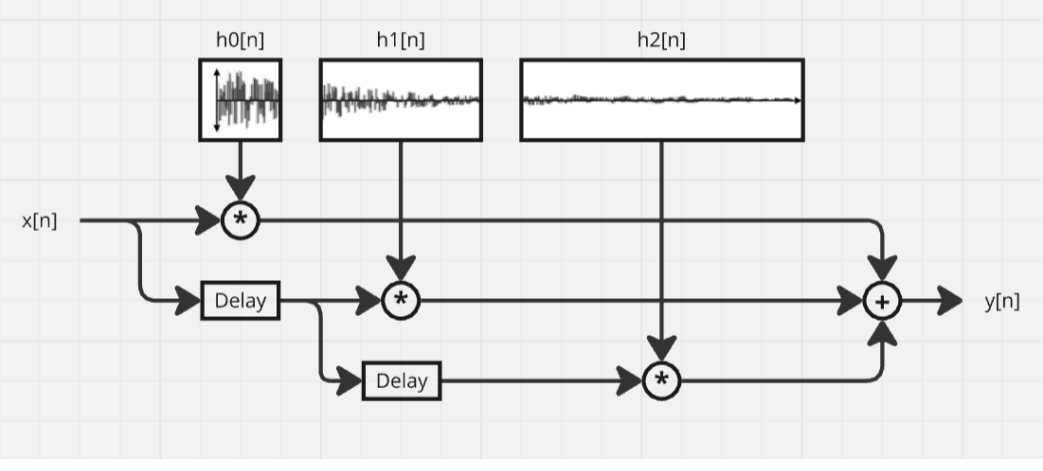
Note: The outputs of later sub-filter stages are not needed right away.

→ They can accommodate more latency without actually delaying the output

→ With more latency, they can handle bigger convolutions

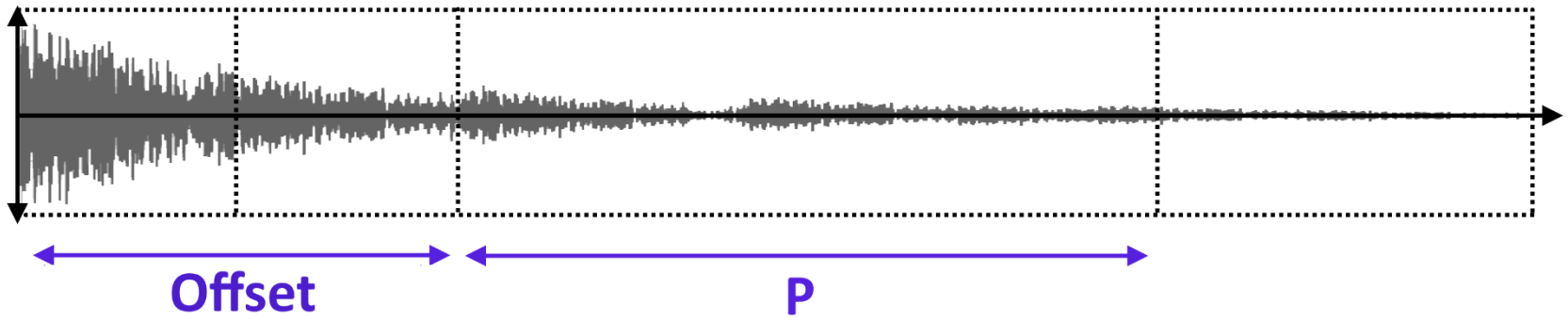
→ Hence the part size can grow as more sub-filters are added





B: Block size
P: Part Size

Clearance constraint



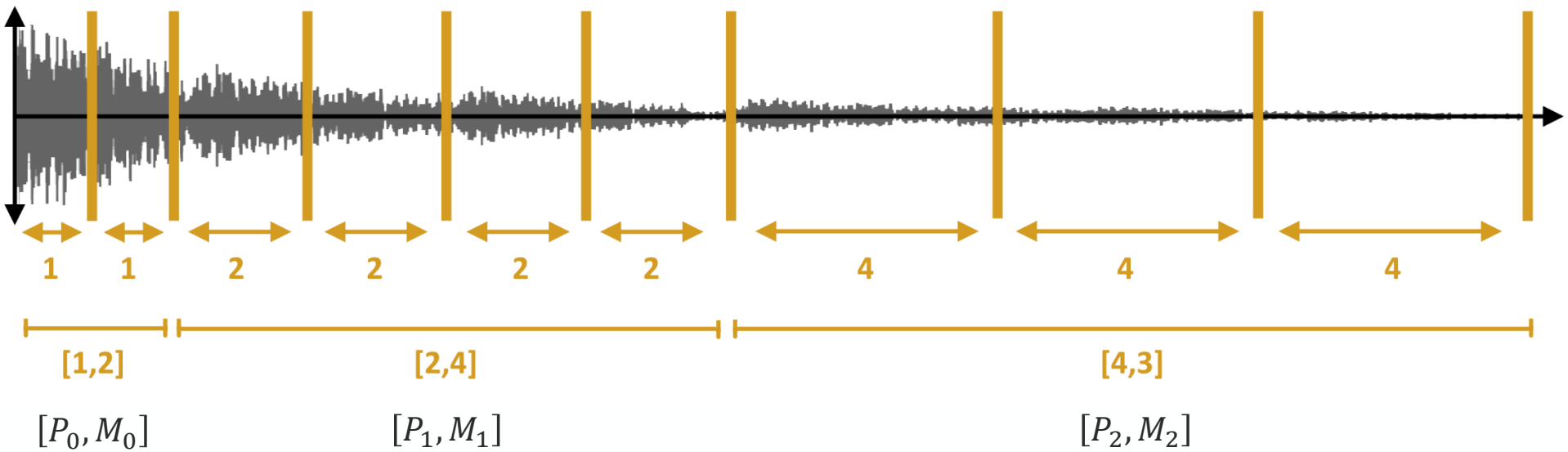
- Each sub-filter must complete its convolution before the result is required for the final sum.
- Therefore, the time taken to accumulate enough blocks cannot exceed the offset.
- The latest block received does not count: it's accumulated in the same cycle period it's used in.

$$P < offset + B$$

$$0 < \underbrace{offset + B - P}_{\text{"Clearance"}}$$

5. Non-Uniform Partitions

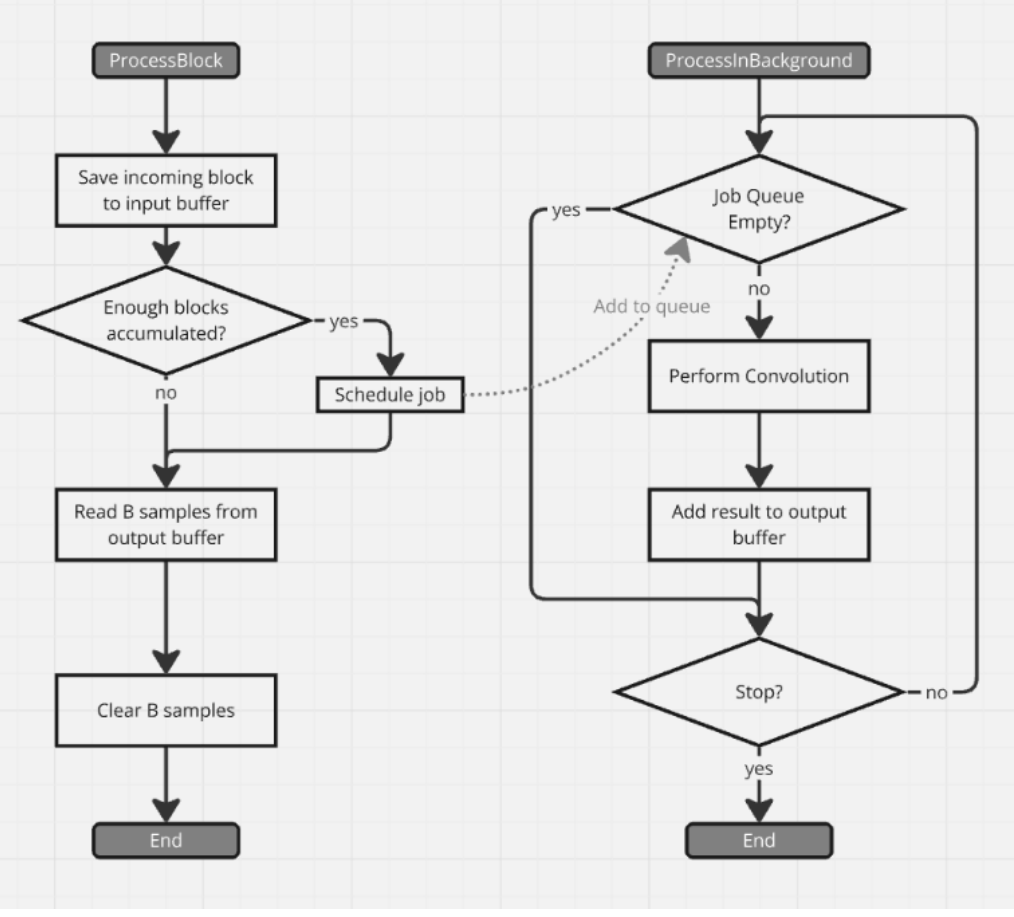
- In theory, the 'best' partition is (1,2,4,8,16,32…): grow the part size as fast as possible while enforcing zero-clearance. This is impractical and only optimal for specific IR sizes.
- Gardner (1995): (1,1,2,2,4,4,8,8,…) + "scheduling", more practical but not perfect
- Garcia (2002): Sub-filters of the same size require the same FFT & IFFT operations. Hence, it's more efficient to group them together as uniform partitions:



Frequency-Domain Convolution

The non-uniformity lets us accumulate blocks (latency) without the typical downsides (delay).
By itself, It doesn't solve the problem of load peaks.

		Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8	Block 9	Block 10	Block 11	Block 12	Block 13	Block 14	Block 15	Block 16
$[8, M_3]$	Stage 4	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Process	Wait	Wait	Wait	Wait	Wait	Wait	Wait	Process
$[4, M_2]$	Stage 3	Wait	Wait	Wait	Process	Wait	Wait	Wait	Process	Wait	Wait	Wait	Process	Wait	Wait	Wait	Process
$[2, M_1]$	Stage 2	Wait	Process	Wait	Process	Wait	Process	Wait	Process	Wait	Process	Wait	Process	Wait	Process	Wait	Process
$[1, M_0]$	Stage 1	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process	Process

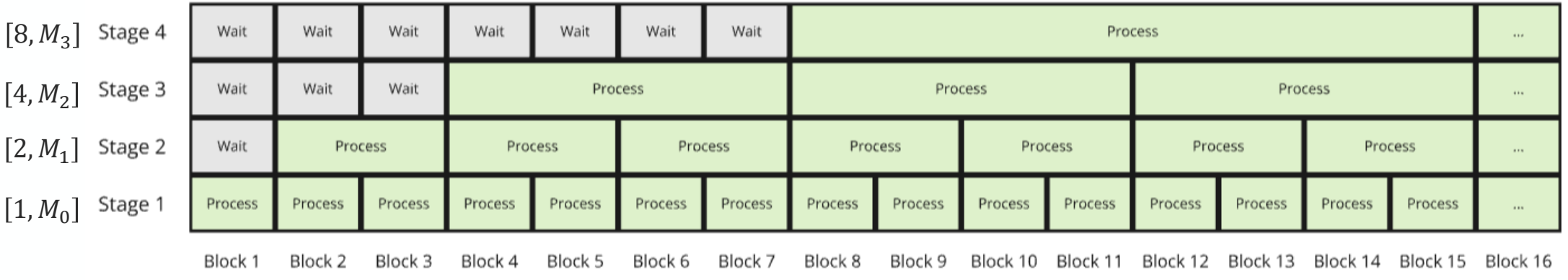


Solution: multi-threading

- The audio thread accumulates and reads blocks, and can handle lightweight, critical convolutions (e.g. the first stage $P_0 = B$)
- The later, big convolutions are computed on separate threads
- As long as they have enough clearance, large sub-filters will finish their work in time.
- We can always increase the minimum clearance to ensure there's spare time.

Frequency-Domain Convolution

Large convolutions have a lot more time than before.
Instead of just waiting for blocks, some work can be done.



Asynchronous processing
(with generous clearances)

5. Non-Uniform Partitions

- Wefers (2012): Generate all valid partitions & find the solution that minimizes the average CPU load.
- For cost estimation, assume each uniform sub-filter is computed on its own thread.
- Do this for various IR Lengths and block sizes to create look-up tables.

Filter length [taps]	**Cost** [FLOPs/ sample]	**Optimal partition**
1024	140.2	$128^{2}\,256^{3}$
2048	163.8	$128^{4}\,512^{3}$
4096	195.9	$128^{4}\,512^{7}$
8192	260.0	$128^{4}\,512^{15}$
16896	304.4	$128^{4}\,512^{4}\,2048^{7}$
33280	344.0	$128^{4}\,512^{8}\,4096^{7}$
41472	360.0	$128^{4}\,512^{8}\,4096^{9}$
53760	384.0	$128^{4}\,512^{8}\,4096^{12}$
66048	408.0	$128^{4}\,512^{8}\,4096^{15}$
70144	416.0	$128^{4}\,512^{8}\,4096^{16}$
82944	439.7	$128^{4}\,512^{17}\,8192^{9}$
100864	451.5	$128^{4}\,512^{4}\,2048^{8}\,16384^{5}$
133632	467.5	$128^{4}\,512^{4}\,2048^{8}\,16384^{7}$
150016	475.5	$128^{4}\,512^{4}\,2048^{8}\,16384^{8}$
201216	498.7	$128^{4}\,512^{8}\,4096^{8}\,32768^{5}$
332288	530.7	$128^{4}\,512^{8}\,4096^{8}\,32768^{9}$
430592	554.7	$128^{4}\,512^{8}\,4096^{8}\,32768^{12}$
524800	578.2	$128^{4}\,512^{8}\,4096^{15}\,65536^{7}$

→ for IRs below 525000 samples, the optimal partition always had:

- At most four different part sizes
- Part sizes ≤ 65536
- Multiplicities ≤ 16

→ We only need 4 uniform sub-filters for most IRs.

→ Approx. twice as fast as Gardner and more practical.

B: Block size
P: Part Size
M: Multiplicity

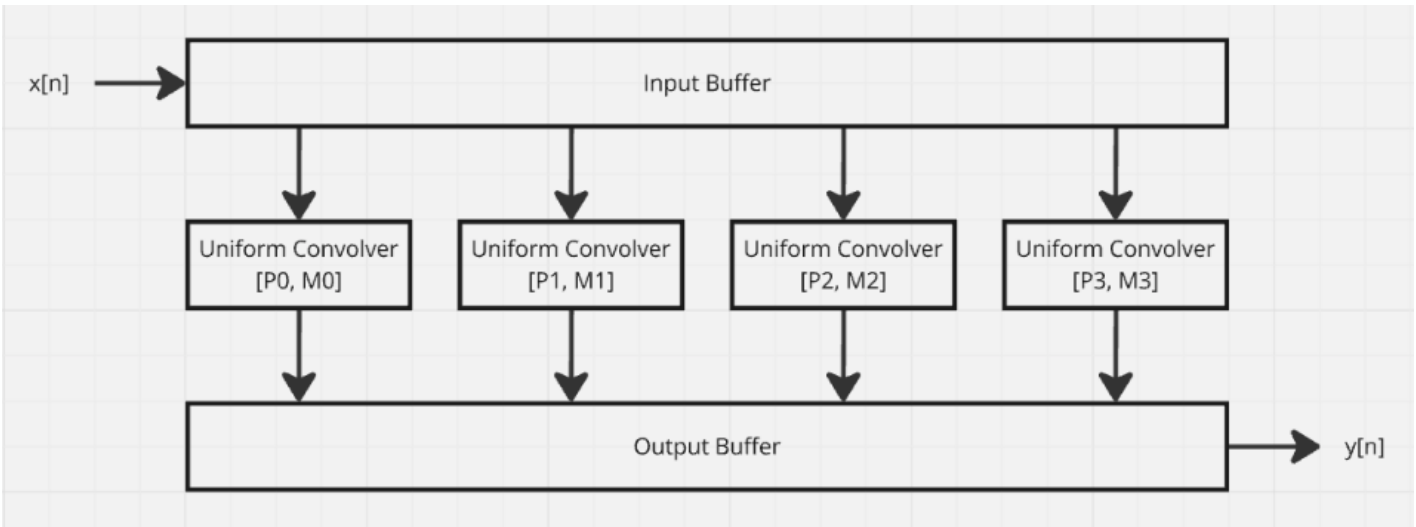
Preparation

Find optimal NUP that satisfies our three constraints.

Initialize four uniform sub-filters with corresponding chunks of the IR.

Processing

1. Save incoming block in the input buffer
2. Whenever $\frac{P_n}{B}$ blocks have been accumulated,
   - Compute the convolution of the uniform sub-filter $[P_n\ M_n]$
   - Sum the result in the output buffer
3. Send B samples from the output buffer to the output stream & clear them
4. Update buffer positions

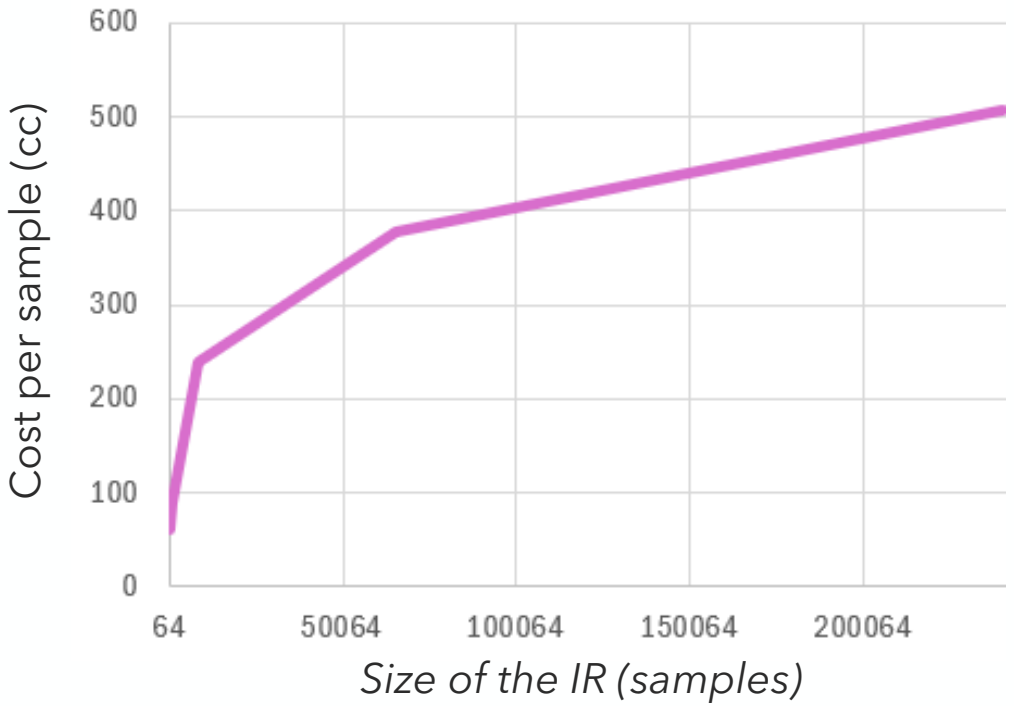


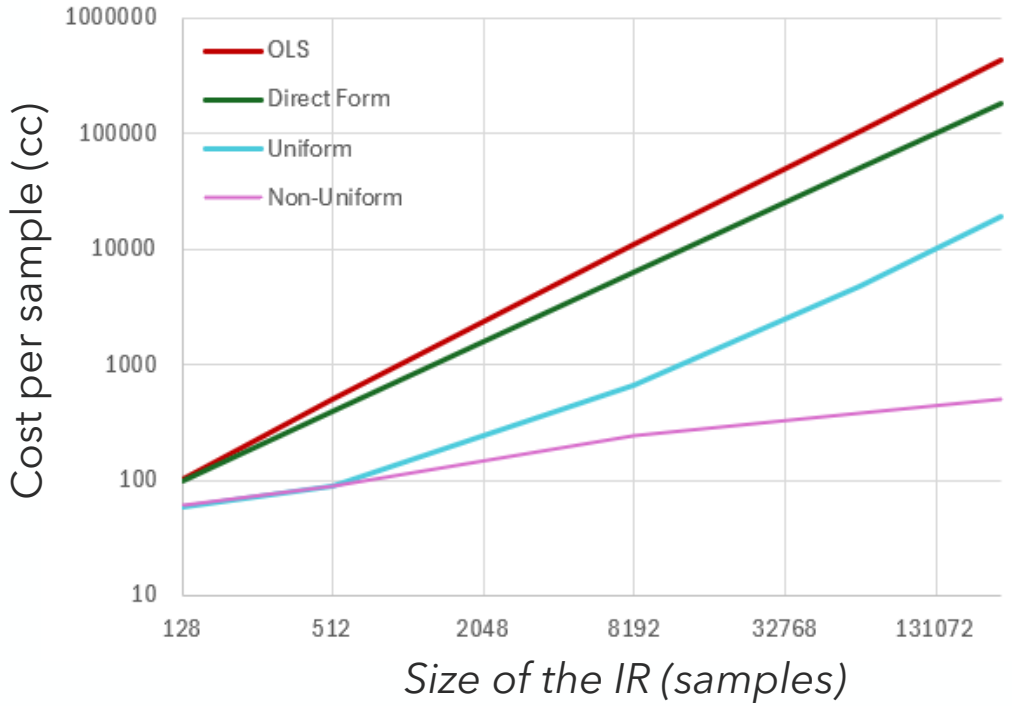
Accumulate blocks

Uniform sub-filters

Combine partial outputs

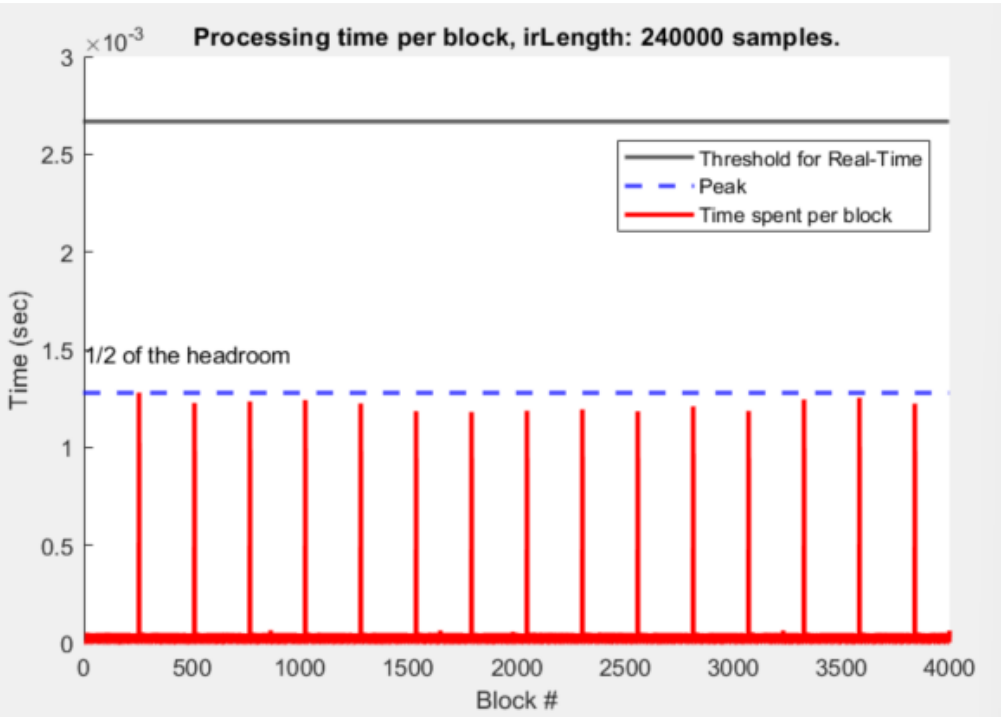
Synchronous processing (single thread)



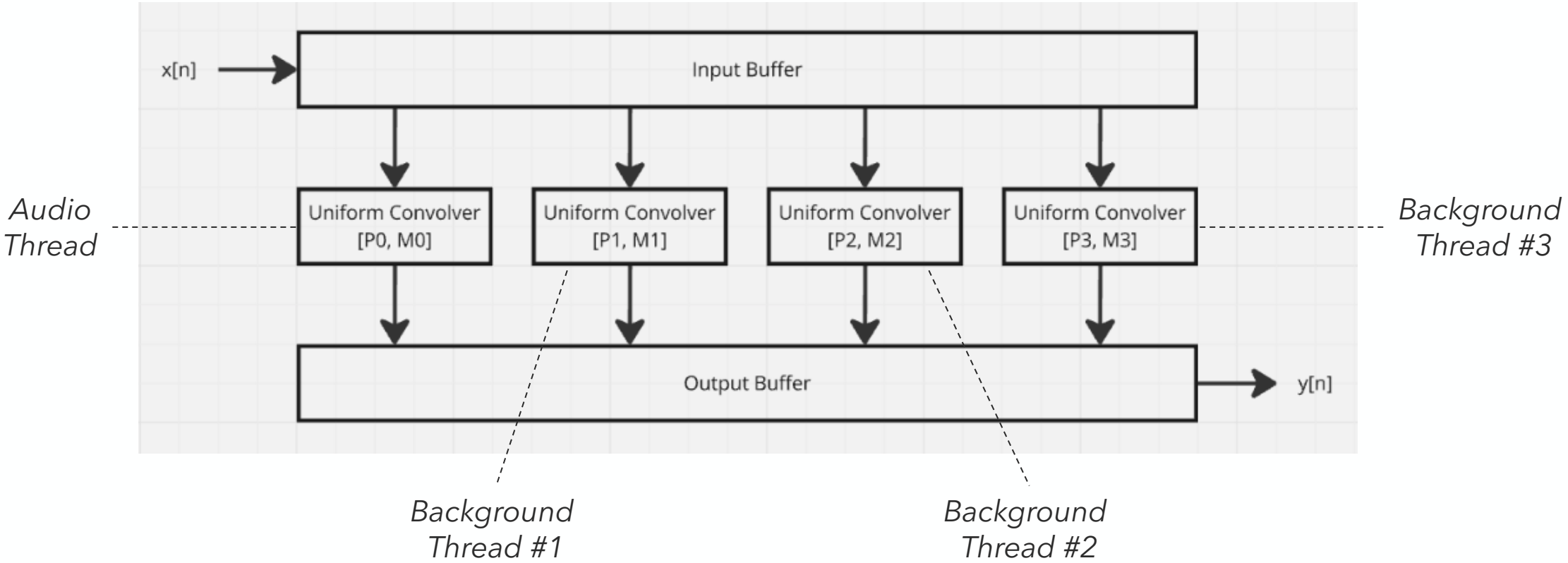


It finally looks logarithmic, but…
Benchmarks only show the average CPU time.

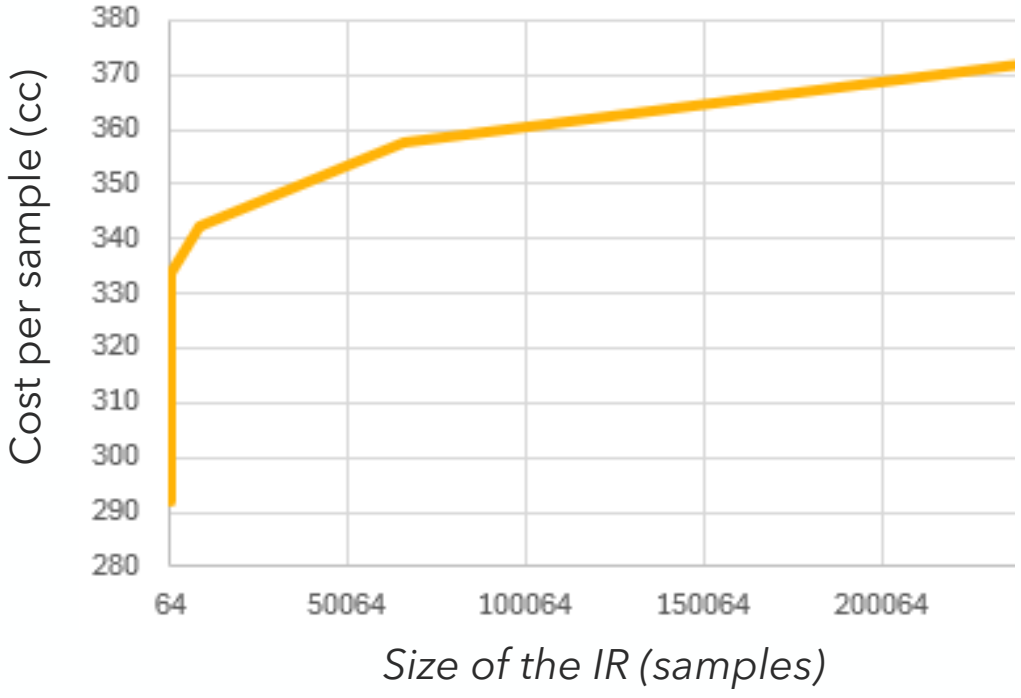
When all the filters are computed on the audio thread, the peak load can become a bottleneck:

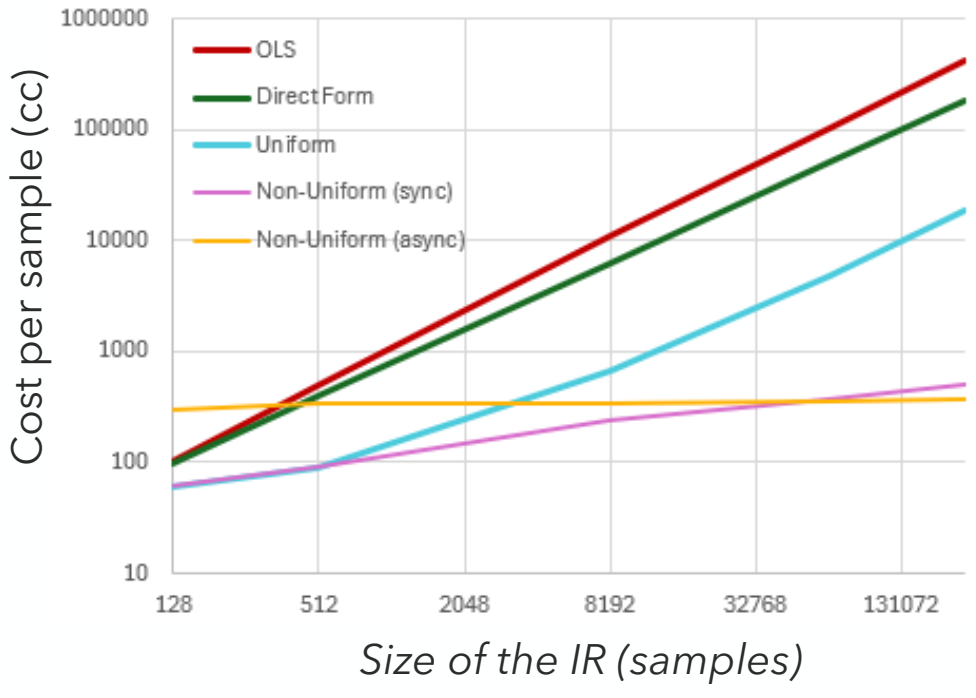


5. Non-Uniform Partitions



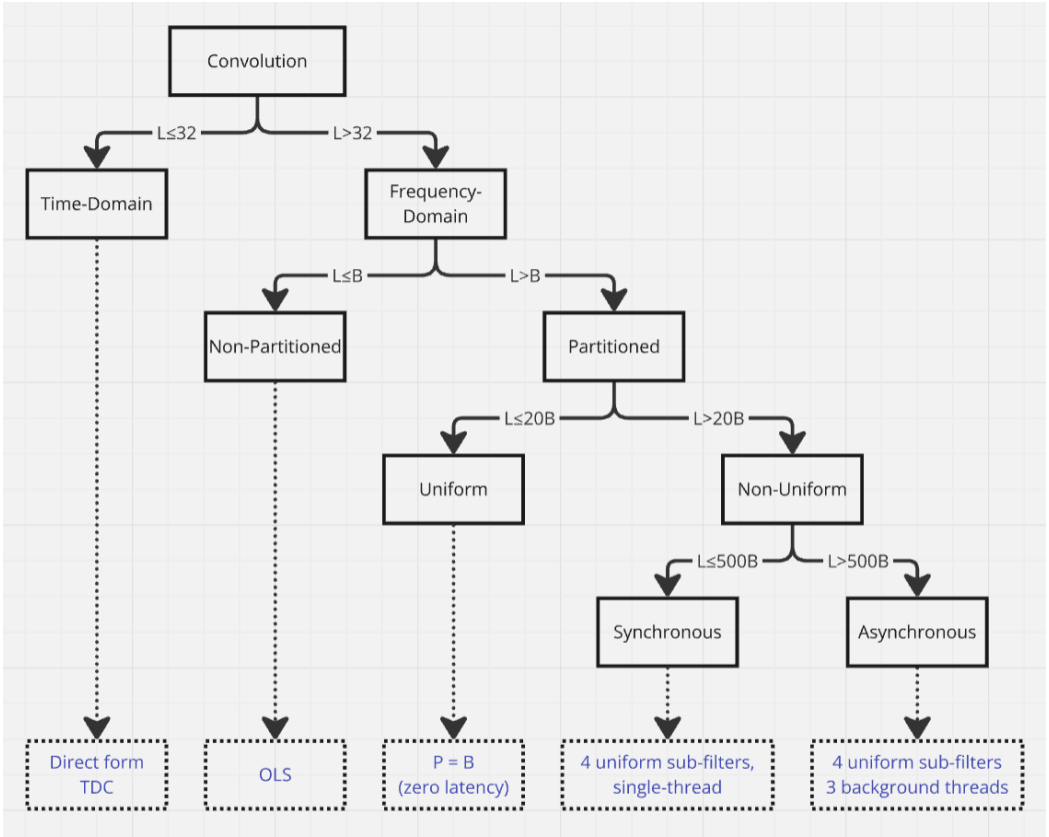
Asynchronous processing (audio thread + 3 background thread)





Conclusion

- TDC still best for very short IRs ($L \leq 32$)
- OLS and OLA by themselves are only useful for specific scenarios ($32 < L \leq B$ or high latency)
- Uniform Partitions are much better and should be able to handle most cases ($\mathrm{B} < L \leq 20B$), even with zero-latency.
- Non-Uniform Partitions can handle long IRs ($20\mathrm{B} < L \leq 500B$). They can be implemented as multiple 'Uniform Stages' in parallel.
- NUPs with additional background threads can handle very long IRs ($L > 500B$)



Summary as a decision tree

Sources and resources

- Stockham (1966) – *High-speed convolution and correlation*
- Gardner (1995) – *Efficient convolution without input-output delay*
- Battenberg & Avizienis (2011) – *Implementing real-time partitioned convolution algorithms on conventional operating systems*
- Wefers & Vorlander (2012) - *Optimal filter partitions for non-uniformly partitioned convolution*
- Wefers (2015) - *Partitioned convolution algorithms for real-time auralization*

Must-Watch

https://www.youtube.com/watch?v=nmgFG7PUHfo
https://www.youtube.com/watch?v=yYEMxqreA10
https://www.youtube.com/watch?v=h7apO7q16V0
https://www.youtube.com/watch?v=spUNpyF58BY

Must-Read

https://ethw.org/Oral-History:James_W._Cooley

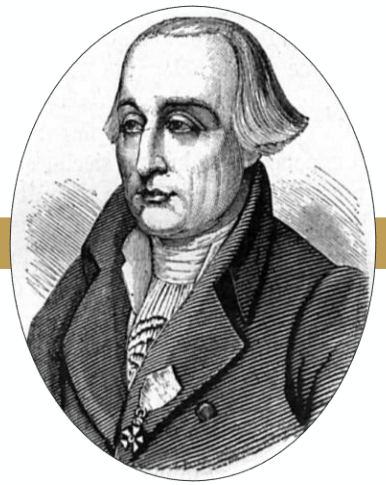
Thank you!

2. Fourier Transform (DFT & FFT)

1740

Euler

- Power series
- $e^{i\theta} = \cos\theta + i\sin\theta$



1750-1770

Lagrange, Clairaut, et. al.

- Trigonometric series
- Discretization

1800-1805

Gauss

- Already using DFT (& FFT!)
- Doesn't publish this work

1807-1822

Fourier

- Solves the heat equation
- Fourier series and theorem

FFT Implementations

	FFTW	IPP	MKL	In-House?
Pros	Popular, well documented	Optimized for intel chips		Fine control
Cons	Expensive license ($7,500), Rarely updated	Only for intel chips		Time-Consuming, Risky

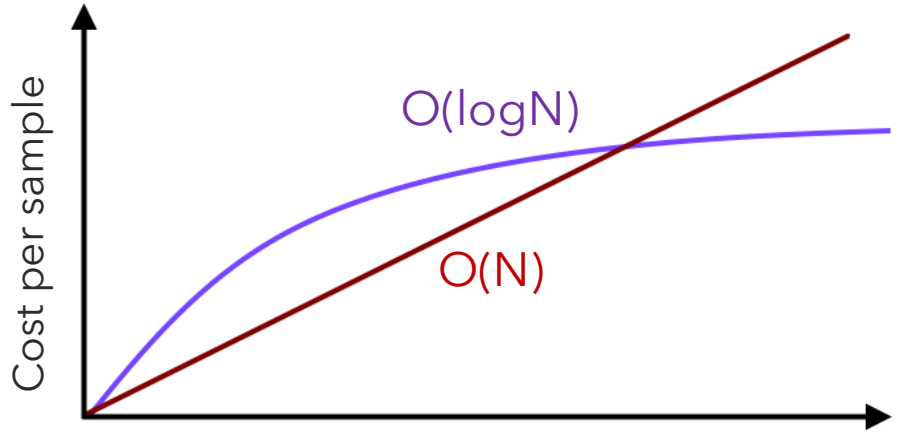
https://github.com/project-gemmi/benchmarking-fft/

```
for (int n = 0; n < blockSize; n++) {
    for (int k = 0; k < irSize; k++) {
        output[n] += input[n-k] * ir[k];
    }
}
```

This is always O(N)

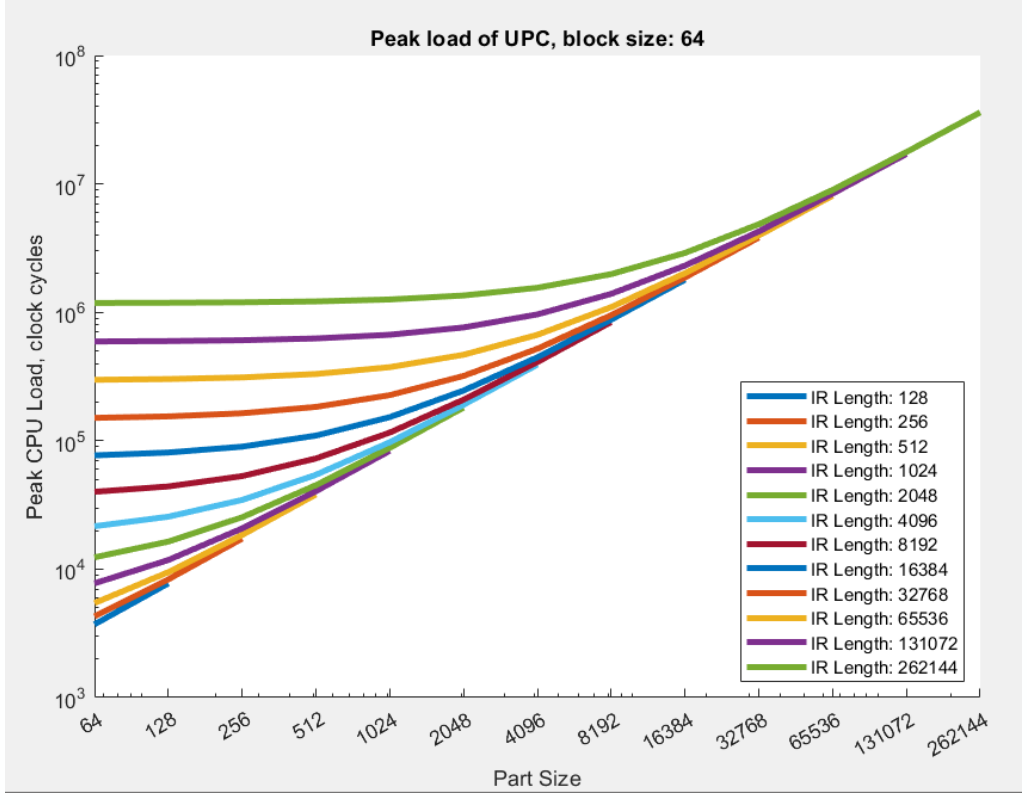
```
inputSpectrum = FFT(input);
for (int i = 0; i < blockSize; i++) {
    outputSpectrum[i] = inputSpectrum[i] * irSpectrum[i];
}
output = IFFT(outputSpectrum);
```

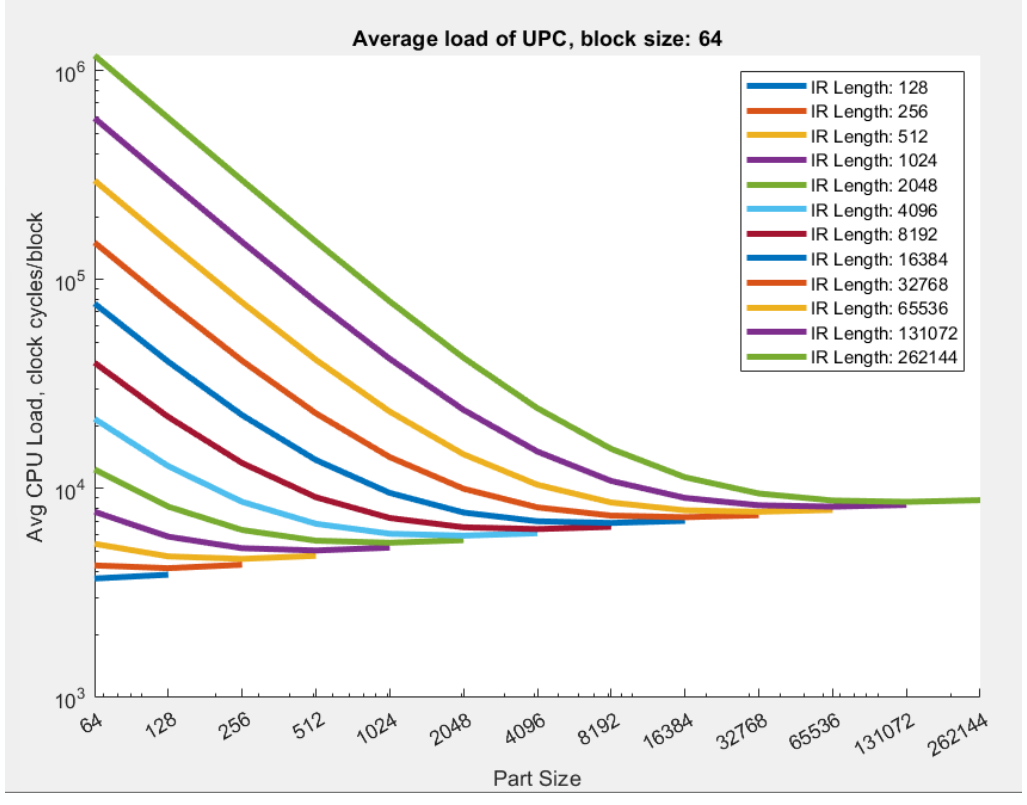
This is O(logN) if irSize ≤ blockSize



Under certain conditions, FDC can be more efficient than TDC for the same result.

Peak and Average CPU load for Uniform Partitions (Estimated)





B: Block size
T: Latency
P: Part Size
N: FFT size (2P)

NUPs offer many degrees of freedom, but there are some constraints:

Constraint #1: Latency

- To extract the most performance, each sub-filter operates at maximum latency: $\mathrm{T} = \frac{P}{B} - 1$
- i.e. Accumulate blocks until they fill up the part size.
- The first sub-filter must have part size $P_0 = B$ to avoid adding actual latency.

Constraint #2: FFT and Part Size

- For each sub-filter, the FFT size N is twice the part size P.
- Assuming the block size B is a power of 2 (32, 64, 128, etc),
- All part sizes must be in the form $P = 2^n B$ to guarantee that N is a power of 2 and a multiple of B.

Clearance (Wefers, 2012)

$$clearance(i) = \sum_{k=1}^{i-1} P_k + B - P_i \geq 0$$

- A sub-filter with positive clearance has spare time to process the convolution.
- With zero-clearance, it must fit the processing within exactly $\frac{P}{B}$ processing cycles
- With negative clearance, it's impossible to compute in time ('non-causal")

	Sync	Async
Clearance constraint	Zero-Clearance	Clearance ∝ P
Pros	Guaranteed output	Can handle very long IRs
Cons	Problematic load peaks for IRs > 1s	Output not guaranteed

Battenberg & Avizienis (2011)

"Ultimately when there are more threads than cores in the system, the responsibility for scheduling the threads falls onto the operating system, which can only do so well given that it has very limited knowledge about the relationships and dependencies between threads."

Multi-tap delay / sparse FIR implemented as a TDC

```
void ProcessBlock(double* input, double* output, int blockSize) {
    // Copy input to internal buffer
    for (int sample = 0; sample < blockSize; sample++) {
        buffer[writePosition] = input[sample];
        // Increment and wrap write position
        if (++writePosition ≥ bufferSize) writePosition = 0;
    }

    // Process taps
    for (int tap = 0; tap < numTaps; tap++) {
        // Retrieve tap info
        int readPos = readPositions[tap];
        double gain = gains[tap];
        for (int sample = 0; sample < blockSize; sample++) {
            output[sample] += buffer[readPos] * gain;
            // Increment and wrap read position
            if (++readPos ≥ bufferSize) readPos = 0;
        }
        // Update read position for next block
        readPositions[tap] = readPos;
    }
}
```

Convolution Reverb